Learning Preference Models for Autonomous Mobile Robots in Complex Domains

David Silver

CMU-RI-TR-10-41

December 2010

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania, 15213

Thesis Committee:
Tony Stentz, chair
Drew Bagnell
David Wettergreen
Larry Matthies

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **DEC 2010**	2. REPORT TYPE	3. DATES COVERED **00-00-2010 to 00-00-2010**
4. TITLE AND SUBTITLE **Learning Preference Models for Autonomous Mobile Robots in Complex Domains**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie MellonUniversity ,Robotics Institute,Pittsburgh,PA,15213**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		

14. ABSTRACT

Achieving robust and reliable autonomous operation even in complex unstructured environments is a central goal of field robotics. As the environments and scenarios to which robots are applied have continued to grow in complexity, so has the challenge of properly defining preferences and tradeoffs between various actions and the terrains they result in traversing. These definitions and parameters encode the desired behavior of the robot; therefore their correctness is of the utmost importance. Current manual approaches to creating and adjusting these preference models and cost functions have proven to be incredibly tedious and time-consuming, while typically not producing optimal results except in the simplest of circumstances. This thesis presents the development and application of machine learning techniques that automate the construction and tuning of preference models within complex mobile robotic systems. Utilizing the framework of inverse optimal control expert examples of robot behavior can be used to construct models that generalize demonstrated preferences and reproduce similar behavior. Novel learning from demonstration approaches are developed that offer the possibility of significantly reducing the amount of human interaction necessary to tune a system, while also improving its final performance. Techniques to account for the inevitability of noisy and imperfect demonstration are presented, along with additional methods for improving the efficiency of expert demonstration and feedback. The effectiveness of these approaches is confirmed through application to several real world domains, such as the interpretation of static and dynamic perceptual data in unstructured environments and the learning of human driving styles and maneuver preferences. Extensive testing and experimentation both in simulation and in the field with multiple mobile robotic systems provides empirical confirmation of superior autonomous performance, with less expert interaction and no hand tuning. These experiments validate the potential applicability of the developed algorithms to a large variety of future mobile robotic systems.

15. SUBJECT TERMS

<table>
<tr><td colspan="3">16. SECURITY CLASSIFICATION OF:</td><td rowspan="2">17. LIMITATION OF ABSTRACT
<b>Same as Report (SAR)</b></td><td rowspan="2">18. NUMBER OF PAGES
<b>182</b></td><td rowspan="2">19a. NAME OF RESPONSIBLE PERSON</td></tr>
<tr><td>a. REPORT
<b>unclassified</b></td><td>b. ABSTRACT
<b>unclassified</b></td><td>c. THIS PAGE
<b>unclassified</b></td></tr>
</table>

Abstract

Achieving robust and reliable autonomous operation even in complex unstructured environments is a central goal of field robotics. As the environments and scenarios to which robots are applied have continued to grow in complexity, so has the challenge of properly defining preferences and tradeoffs between various actions and the terrains they result in traversing. These definitions and parameters encode the desired behavior of the robot; therefore their correctness is of the utmost importance. Current manual approaches to creating and adjusting these preference models and cost functions have proven to be incredibly tedious and time-consuming, while typically not producing optimal results except in the simplest of circumstances.

This thesis presents the development and application of machine learning techniques that automate the construction and tuning of preference models within complex mobile robotic systems. Utilizing the framework of inverse optimal control, expert examples of robot behavior can be used to construct models that generalize demonstrated preferences and reproduce similar behavior. Novel learning from demonstration approaches are developed that offer the possibility of significantly reducing the amount of human interaction necessary to tune a system, while also improving its final performance. Techniques to account for the inevitability of noisy and imperfect demonstration are presented, along with additional methods for improving the efficiency of expert demonstration and feedback.

The effectiveness of these approaches is confirmed through application to several real world domains, such as the interpretation of static and dynamic perceptual data in unstructured environments and the learning of human driving styles and maneuver preferences. Extensive testing and experimentation both in simulation and in the field with multiple mobile robotic systems provides empirical confirmation of superior autonomous performance, with less expert interaction and no hand tuning. These experiments validate the potential applicability of the developed algorithms to a large variety of future mobile robotic systems.

Keywords: Mobile Robots, Field Robotics, Learning from Demonstration, Imitation Learning, Inverse Optimal Control, Active Learning, Preference Models, Cost Functions, Parameter Tuning

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The field of robotics is poised to have a dramatic impact on many industrial, scientific, and military domains. Already, complex robotic systems are moving pallets in warehouses, dismantling bombs or finding mines in volatile regions, and searching for water on Mars. These applications demonstrate the capability of robotic systems to increase efficiency, improve safety, or perform tasks that would otherwise be prohibitively expensive. In the near future, the size and scope of robotic applications is sure to continue expanding.

Unfortunately, in many regards robotics still remains an experimental field. Building robots to perform simple, well structured tasks a limited number of times is a sufficiently simple goal to be frequently achieved by elementary school students. However, when robots must interact with the real world and demonstrate high reliability, the challenges are greatly magnified. Under these circumstances, constructing and fielding robotic systems still requires an enormous initial investment of resources and labor in order to engineer a system capable of operating in real world domains. Despite this high cost, such systems often still require a high degree of human monitoring and maintenance to ensure that they continue to perform their tasks.

This document focuses on some of these problems that are inherent in developing and fielding autonomous robotic systems for challenging, unstructured and dynamic environments. Specifically, the problem of constructing and tuning preference models in mobile robotic systems is explored. Automated techniques are proposed, developed and demonstrated to address both the inefficiency and sub-optimality of traditional labor-intensive approaches to hand tuning system behavior, which involve large amounts of manual engineering and expert interaction. This document is structured as follows: this chapter introduces some of the basic challenges of autonomous mobile robotics. Chapter 2 sketches a brief outline of recent mobile robotics research and development. Chapter 3 analyzes this overview and identifies the problem of preference model construction as central to the continuing development of autonomous mobile robotics. Chapter 4 explores different possible approaches for tackling this class of problem, before deciding on the idea of Inverse Optimal Control based Learning from Demonstration. Chapter 5 describes the Maximum Margin Planning framework and the associated LEARCH, R-LEARCH, and DR-LEARCH algorithms for learning preferences over terrain types for mobile robot perception, with Chapter 6 extending this approach to the PD-LEARCH algorithm for learning driving styles and preferences from expert demonstration. Additional issues relating to the efficiency, stability and consistency of such learning frameworks are discussed and addressed in Chapter 7, and the document concludes with a discussion and directions for future work in Chapter 8.

1.1 Mobile Robotic Systems

Mobile robotic systems have an enormous potential to benefit society. In many cases, robotic systems (either experimental or hardened) have already undergone initial deployment. An application with which many are now familiar is that of simple mobile systems for performing coverage tasks, such as cleaning floors or mowing lawns. These systems have the advantage of requiring relatively little intelligence or sensing, and therefore are capable of being priced for the individual consumer market.

Robotic systems have long been used in industrial settings. However, the industrial application of robotics is beginning to grow beyond the automation of assembly lines and other high precision tasks. For instance, large warehouses and storage facilities are beginning to integrate mobile robotic systems for moving pallets and other large or heavy items. As a consequence, the logistics of such sites are greatly simplified. The deployment of these systems is beginning to fulfill the promise of efficient robotic load handling that has already been briefly glimpsed in automated container handling [1]. In addition to industrial applications, mobile robotic systems have begun to make their way into the mining industry. Experimental systems for performing Load-Haul-Dump (LHD) operations have been deployed for over a decade [2]. In addition, systems are also under development to aid in surface mining operations from haul trucks to large excavators [3]. Another industrial application of mobile robots is in the area of inspection and mapping. In this application, robots have already touched the traditional 3 D's of robotics applications; mobile systems have handled dirty jobs such as pipe and sewer inspection [4], dangerous jobs such as probing a nuclear reactor or mapping abandoned mines [5], and dull jobs such as large scale survey and mapping.

Mobile robots are also being applied to the office setting. An initial deployment success has been in hospitals, many of which now routinely use robots to deliver supplies from central storage to individual rooms, freeing personnel to deal with actual medical issues. The broader scientific community has also begun to make use of robotic systems. Mobile systems have made contributions ranging from exploration of volcanoes [6] to the depths of the ocean [7]. Perhaps the greatest success in this area has undoubtedly been the NASA MER missions to Mars, which have allowed remote exploration and geology to take place for over 6 years (including the recent addition of autonomous target selection for scientific study [8]).

The single field that has seen the most recent impact due to robotics is the military domain. The use of Unmanned aerial vehicles for both extended reconnaissance and precision strikes has grown rapidly. Ground systems such as the Packbot and Talon have become an essential tool for detecting and disarming IEDs in Iraq and Afghanistan [9]. These system have to date been mostly tele-operated, which is one reason for their effectiveness and popularity.

1.2 Real World Challenges of Robotic Systems

Despite the preceding list of actual applications and varying degrees of success, most mobile robotic systems are not yet ready for mainstream adaptation. The true killer applications of mobile robotics, such as the generic self driving vehicle, remain out of reach. In an attempt to diagnose what is still lacking that prevents such deployments, it is useful to investigate some of the common properties of the previously described successes:

Figure 1.1: Examples of deployed mobile robotic systems

Custom, Highly-Engineered Solutions All of the systems were designed for very specific applications and scenarios. To some degree, this is inevitable; one wouldn't attempt to use a robot designed for aerial reconnaissance in a sewer inspection task. Each proposed task naturally comes with its own requirements and constraints in terms of the size, shape, mobility, effectors etc. of the robotic system. However, this specificity can also be used as a crutch. A robot that performs a LHD task in a mine could potentially move pallets in a warehouse with just a change of end effectors. However, by designing a robotic system to be used only in specific scenarios and environments, numerous simplifying assumptions can be made to reduce the complexity of the system's software.

Structured or Semi-Structured Environments In addition to assumptions about the tasks to be performed, many of these systems make explicit assumptions about the environments in which they operate. In some cases, the environment must be explicity structured **for** the robot (e.g. warehouse transport). Other systems do not require explicit engineering of their operating environment as long as it conforms to certain standards (e.g. supply delivery in hospitals). Still other systems can tolerate some degree of ambiguity in the definition of their operating environment, as long as certain key assumptions still hold (e.g. home coverage tasks or underground LHD).

Varying Degrees of Manual Control All these robotic systems require some level of human input. In some cases, input consists only of high level instructions (clean this room, begin daily deliveries, etc.). In other cases, individual tasks are continually specified by a human operator (drive to a certain location, move a pallet to a specific workstation, etc.). The extreme end of this spectrum is direct tele-operation, where a human operator specifies all of the robot's motion commands.

Low Sensitivity to Robot Failure Any complex system is likely to have a number of possible failure modes, and robots are no exception. The acceptable failure rate for a specific application is one of the factors determining when a robotic system becomes deployable. Many current applications have a very low sensitivity to a failure by the robotic system. If a robot vacuum cleaner becomes stuck and shuts off, floor cleaning is simply delayed by a day. If a robot in a warehouse shuts down, the flow of pallets is temporarily slowed; in a pinch a human in a fork-lift can substitute. While all of these scenarios are undesirable and would result in a certain degree of annoyance, none of them could be said to result in the failure of a mission critical system.

These common features of the mobile systems that are actually field deployed provide clues as to the key challenges that remain to be solved before truly wide real-world deployments will be possible:

Unstructured Environments In general, no permanent assumptions can be made about the appearance or structure of the world. Not all environments can be easily classified into navigable freespace and obstacles, or easily partitioned into sets of objects with semantic meaning. Further, this lack of assumed structure results in the requirement of a higher signal to noise ratio in perception and world modeling, in order to properly perceive the meaningful structure of the environment. Therefore, the added complexity of unstructured or un-engineered environments necessitates higher resolution sensors, sensor processing, and modeling capabilities in mobile systems.

Dynamic and Partially Known Environments The world is constantly changing, and any observation that is not current has a chance of being stale and incorrect. Further, not every location is constantly observable. Interaction with and modeling of unobserved or partially observed terrains or locations is almost always necessary.

Unpredictable Environments and Reactions The world is not deterministic. Every action has a degree of unpredictability. A patch of floor that was dry several minutes ago may now be wet, impacting mobility. Or a mine corridor that is reported as clear may suddenly become blocked. Robotic systems must be able to account for such uncertainty in order to interact with a nondeterministic world.

Mission-Critical Scenarios Some potential robotic tasks are truly mission-critical, with an exceptionally high cost of failure. In the case of a robotic car or a fully autonomous armed robot, a mistake may literally be fatal. Therefore in some scenarios, due to moral and legal reasons, a mobile robotic system must not just meet but far exceed human safety and dependability before adoption can be mainstream.

Full Autonomy The ultimate goal of many mobile robots is to perform some task in lieu of a person all the time, not some of the time or with the continual aid of a person. If the desire for a mobile system is driven by any requirement aside from tele-presence, then mobile systems must become capable of operating without any but the highest level and lowest frequency of human input in order to become truly ubiquitous.

Development Effort Building a mobile system that conquers the preceding challenges is difficult enough. Unfortunately, the cost of development must also be considered. A significant manual engineering effort is often necessary, consisting not only of an initial design and development but also a considerable amount of system tuning and tweaking. Additionally, system testing and validation can also occupy a significant amount of time and other resources; in some cases even more than the initial design and development.

A common thread amongst these challenges is the requirement that a mobile system must be autonomously capable of the correct behavior under vague and complex circumstances. The difficulty in fulfilling this requirement lies in the intricacy necessary to properly encode a mobile system with the ability to generate correct behaviors, along with the ability to model and understand the correct behavior under particular circumstances.

Figure 1.2: Examples of experimental mobile robotic systems

1.3 Real World Requirements of Autonomous Systems

With a clear picture of the challenges that await any widely deployed mobile system, it is now possible to define high level requirements for such a system. That is, in order to claim that the preceding challenges have been overcome, autonomous mobile systems will require at least the following properties

Reliable The system must have a sufficiently high rate of success, mean time to failure, etc. Defining reliability is tricky, as it is a relative measure depending on the form and price of failure. If a robot fails to pick up a pallet that is a very low cost failure in comparison to a robot crashing that pallet into a wall. However, reliability can also be defined in terms of confidence in the system. For example, the braking system of a car is not 100% reliable, and yet it is rarely something that is thought about at the start of a ride. Therefore, it is sufficiently reliable that we assume it will work, and if that assumption proves faulty the consequences are dealt with. Continuing this analogy, mobile robotic systems must become sufficiently reliable that rather than requiring monitoring in preparation for a failure, they are simply left to their own devices; the cost and probability of a failure are such that it is better to respond to the consequences of a failure after the fact.

Robust The system must perform well in the face of uncertain or novel scenarios. Robustness and reliability are partially coupled; a system that is not sufficiently robust will almost certainly not be sufficiently reliable. However, there is more to robustness than simply reliably handling rare circumstances. Again, it is a question of confidence. For a system to truly be considered robust, there must exist sufficient confidence that it will perform reliably in completely novel circumstances (including scenarios or inputs for which it has never been tested) or that performance will degrade gracefully. In the same way sufficiently reliable can be defined as when constant monitoring of the system is no longer required, sufficiently robust can be defined as when monitoring of the environment is no longer required.

Rapid Development Mobile systems must be simpler and faster to design and implement. This does not imply that the overall complexity or power of the systems must decrease. Rather, the human resources requirement must be reduced. Specifically, the amount of time from system design and development to final deployment must be reduced. Currently, this period is occupied by enormous amounts of system tweaking and redesign, parameter tuning and optimization, and numerous rounds of validation testing, and can go on for years. Again, this requirement is somewhat relative and depends on the complexity of the system and its purpose, but fundamentally the proportion of time spent on system optimization and validation must decrease.

Reusable Mobile systems must be increasingly built using off the shelf components. Great strides have already been made with respect to hardware; many mobile systems now use exclusively off the shelf computing and sensors. However, most software components are still custom built, even when the component simply re-implements standard algorithms or heuristics. In some regards, this is more a software engineering than a systems or robotics engineering issue, and is a problem shared with many other disciplines. However, where robotics differs is in the latter stages of system development. Initial designs for mobile robotic systems often make use of off the shelf algorithms if not off the shelf software. However, later stages of system development often result in new components or parameters that are very platform or task specific. Usually, this is in response to system failures or shortcomings demonstrated through early testing. As previously discussed, a great deal of engineering resources are often spent during this period; even worse, most of these resources are spent in such a way as to have no further benefit to any future systems. It would be useful if not only software components but the results of tuning such components were reusable and could generalize across systems and operating conditions.

These four requirements encapsulate much of the work that remains before mobile systems are widely deployable. The next Chapter will explore previous work in field deployed mobile robotic systems, and Chapter 3 will discuss how well these systems meet these requirements.

Chapter 2

Related Work in Autonomous Mobile Robotic Systems

The field of mobile robotics is often partitioned into systems designed for either indoor or outdoor environments. These two scenarios each present a very distinct set of challenges and requirements, and often result in very different solutions for both software and hardware. However, in many ways this distinction between indoor and outdoor mobile robots is not optimal. For instance, some robots that operate indoors must still solve complex perception tasks that are more common in outdoor settings, and some outdoor systems operate in simplified or engineered environments that are more similar to indoors. Therefore, it is more useful to use the following taxonomy when describing the operating environment of a mobile system

Structured Environments: Very strong assumptions can be made about properties of the environment. Environments that can be reliably partitioned purely into navigable freespace and obstacles are a classic example. Most indoor environments fall into this category, as would an engineered outdoor environment consisting only of flat ground and positive obstacles. A high level of environmental engineering is implied; that is the environment will be modified to enforce any assumptions made about it.

Semi-Structured Environments: Some assumptions can be made about the environment. Urban environments are a classic example: it is assumed that there are certain proper places to drive (roads) that are easy to detect but may be occupied by dynamic obstacles. Underground mines are another good example: the overall tunnel structure of the mine is taken as a given, but debris or equipment may also be present. A moderate level of environmental engineering is implied. It is often the case that the environment was originally (at least in part) artificially constructed. However, continuous modification to enforce strong environmental assumptions does not occur.

Unstructured Environments: No assumptions can be made about the environment. True cross-country traverses are a good example: while roads or flat areas may be present, they can not be assumed to exist. Negative obstacles and other difficult to detect hazards are a possibility. Positive obstacles may consist of rigid objects or of easily traversable vegetation. A low to nonexistent level of environmental engineering is implied; in any case none can be assumed to have taken place.

Unlike the indoor/outdoor distinction, this categorization is not strictly well defined. However, it remains useful as a method for indicating how complex a world model will be necessary for operating in a given environment. Structured environments require simple world models; any part of the environment that is too complex can be modified to fit assumptions about structure. In contrast, unstructured environments require complex world models; as no assumptions are made about the environment, any property of the terrain that would affect a robot's decisions or interactions must be modeled.

In this document, mobile robots in structured environments are not discussed or explicitly addressed. This is not to say that the problem of autonomy in such environments is 'solved' by any means. However, the state of the art in such scenarios is significantly more advanced, due to both the structure of the environment, and the constraints that places on behavior. Instead, this document is concerned with the challenges inherent in understanding and interpreting semi and unstructured environments, as well as making decisions in such environments. What follows is a brief (and by no means complete) chronological history of research for robotic systems in such environments, with the goal of fashioning an understanding of the current state of the art, as well as what problems and trends are developing and where this will lead research in the future. Prior work in mobile systems operating in semi-structured environments are described first, followed by systems designed for unstructured environments in Section 2.2. The focus of these sections is on wheeled or tracked autonomous mobile robots , and the perception and motion planning challenges inherent in such robots. Systems that are primarily tele-operated or rely heavily on tele-operation are not discussed. The higher dimensional planning and control problems inherent in legged locomotion are also outside the scope of this work (although similar approaches have been applied with success [10]).

2.1 Mobile Systems in Semi-Structured Environments

The first mobile robot to operate outdoors was the Stanford Cart [11, 12]. While exceedingly simple by current standards, the Cart pioneered many of the technologies still used today. A Sense-Plan-Act architecture was utilized, with stereo vision as the single mode of both perception and positioning. Motion plans were computed using an approach similar to visibility graphs, computed on a configuration-space expansion of perceived obstacles. A single Sense-Plan-Act cycle took 10-15 minutes, and would result in approximately 1m of traverse. The longest single run was 20m. Despite this simplicity, many of the problems encountered by the Cart while operating outdoors are still challenges to modern mobile systems. For instance, the stereo vision perception system had difficulty finding sufficient visual features to locate and track; the assumption of a sufficiently feature rich environment that held indoors did not hold outdoors. Additionally, a static environment assumption no longer held, and even slow dynamics such as shadows moving due to the sun's position could effect the system. Finally obstacles were defined as those features which were “sufficiently above” the floor, with the floor being computed from a planar ground assumption. While seemingly simple to define, the definition of “sufficiently above” is an early example of system a parameter that could drastically affect the Cart's behavior if not properly tuned.

The Terregator mobile robot [13] was the first robotic platform designed specifically for operating in more complex and potentially off-road outdoor environments. Over the years, multiple autonomy systems were developed on top of Terregator, including a system for following roads and sidewalks [14], a system for the exploration and navigation of mine corridors, and a system for the navigation of more complex off-road terrains [15, 16]. The latter of these modes of operation is most relevant to the challenges faced by modern robots. In this mode, 3D range data was used to classify terrain as either explicitly passable (smooth and flat) or possibly not passable (obstacles or rough terrain). However, even this seemingly simple distinction could result in unexpected behavior, as sometimes flat off-road terrain such as grass would be preferred to roads and paths when potholes were present. Additionally, this conservative classification would sometimes result in no surrounding terrain appearing as passable, resulting in an autonomy failure even when Terregator was physically capable of continuing to traverse (obstacles up to 15 cm in height could be surmounted).

Early work on the Navlab vehicle [17] continued the work of Terregator and combined road following and obstacle avoidance capabilities, while adding robustness to real world operating conditions such as changing weather and lighting, or dynamic obstacles [18]. In addition to operating in on road environments, simple off road operation was also possible. 3D Range images would be processed to identify 'inadmissible' regions of terrain [19]. These regions were identified by analyzing the terrain geometry for regions which would violate constraints on vehicle tilt, ground clearance, or a minimum amount of contact between wheels and the support surface. This demonstrates one of the key assumptions that often helps to distinguish systems for structured and semi-structured environments: the assumption that the visible geometry of a scene represents something close to the supporting ground surface.

Also during this time, a great deal of work took place in Europe under the umbrella of the PROMETHEUS project that focused primarily on purely vision based approaches for autonomous on road vehicles. The result was the VaMoRs and VaMP vehicles [20, 21] which were capable of driving in actual traffic conditions at highway speeds. Computer vision algorithms and hardware was used to identify lane markings as well as other vehicles, allowing for both lateral and velocity control for highway operation. In later stages of development range sensors were integrated for distance keeping in traffic. Along with the goal of producing fully autonomous systems, there was also work to develop component sensing technology that could be integrated into consumer vehicles.

The early to mid nineties continued to see development in vehicles designed to autonomously detect and track lane markings for highway driving [22]. By 1996, there were two large demonstrations of the potential of such approaches. The “No Hands Across America” demonstration [23, 24] involved Navlab traveling nearly 5000 km, operating autonomously 98% of the time. The VaMP vehicle also completed a demonstration of 1600 km of travel, operating autonomously 95% of the time. Both demonstrations took place at actual highway speeds, demonstrating reliability and robustness under real world operating conditions. The next decade saw several efforts to continue to improve this performance, as well as extending operation to urban driving [25, 26, 27, 28, 29]. Along with lane tracking and collision avoidance, this required the additional integration of pedestrian tracking and road sign detection capabilities, along with the capability to plan intelligently and reasonable given the unpredictable nature of the environment being observed.

In 2005, the DARPA Grand Challenge produced a number of high performance systems for navigating long distances in semi-structured environments. There is some debate as to whether the Grand Challenge took place in unstructured terrain or not. Based on the previously described definitions, the Grand Challenge should be considered as a navigation task in semi-structured terrain for the following reasons

- The race took place on unpaved roads. Although more difficult than paved roads, unpaved roads are still specifically constructed for ground vehicle mobility
- There were no significant negative obstacles. Positive obstacles could always be avoided, as it was assumed that the route contained reasonably flat ground to traverse [1]
- The required route was well defined and constrained a priori through GPS waypoints, and in some cases was later modified by hand.

Regardless of this distinction, the Grand Challenge was an enormous step forward in reliable autonomous systems, as 5 vehicles successfully completed the 220 km (132 mile) course without human aid, with top average speeds of almost 9 m/s (32kph or 19 mph) [30, 31, 32, 33]. These vehicles serve as further demonstrations of the coupling between robustness and reliability. Despite issues with drive by wire systems, individual sensors, or even temporary failure of software processes, these vehicles were able to gracefully recover without human aid and complete their task.

The successful systems had many properties in common. Their perception systems were primarily range data driven (usually gathered via LiDAR). Although the details varied, geometric properties of the collected range data were used to identify driveable and non-driveable regions of the local environment. Various local motion planning approaches were used to produce safe and feasible actions. Velocity control was also taken into account, trading off the maximization of safety with the minimization of race time. Finally, the systems were heavily tested, with some teams reporting thousands of kilometers of full system testing [30, 31].

One of the most important results of the Grand Challenge was the demonstration that perception can be robust in general, even in the face of semi-structured terrain, changing weather and lighting conditions, dust, and other sources of noise in calibration and positioning. Notable in this regard was the Stanford racing team. Stanford used supervised learning to map from range data to obstacle classifications [30], as opposed to manually tuning and adjusting this mapping. This resulted in a more robust classification model, that was robust even to errors in other components of their system (such as positioning). This demonstrates that sometimes it is easier to train a system than to manually tune it. Additionally, Stanford also used online near-to-far learning, with the system learning online to interpret camera images from processed range data [30, 34]. Adaptive online learning was also used by the CMU team to predict the presence of roads just outside of LiDAR range [31]. From a systems standpoint, what is most notable is that adaptive components were trusted as part of a mission critical system. This demonstrates that learning algorithms and modules, once properly verified, validated and characterized, do not require human monitoring any more so than manually engineered algorithms and components.

Perhaps as important as the successful completion of the Grand Challenge was the demonstration of just how difficult the necessary level of reliability was to attain. Nearly 200 teams originally petitioned to enter the competition, with only 40 attending the qualifier event and 23 participating in the final race. Of the 18 teams that participated but did not finish, the average distance to failure was approximately 42 km (25 miles), or less than 20% of the full course length [35]. The causes of failure ranged from mechanical and hardware failures to both previously known and unknown

[1] This assertion is backed up by the fact that many of the teams, including the winning vehicle, classified terrain in a binary manner

software bugs or algorithmic issues. These failures serve to demonstrate an important point about reliability and robustness, namely that individual component reliability is necessary but not sufficient. Instead, not only must each system component behave properly, but the manner in which components are coupled must produce a reliable system as well.

In 2007, the DARPA Urban Challenge served as another incentive for the development of high quality mobile robotic systems. The attrition rate was similar to the Grand Challenge: 89 teams petitioned to participate, 35 were invited to the qualifier, 11 participated in the final race, and 6 vehicles [36, 37, 38, 39, 40, 41] successfully completed the 96 km urban course, with a top average speed of just over 22 km/h. The successful teams all had several components in common: perception systems that could detect not only static but track dynamic obstacles, multiple layers of planning systems for both mission level and local motion planning, and higher level reasoning to make decisions regarding passing, merging, yielding, etc.

Unlike in the Grand Challenge setting, urban environments are engineered to provide a much clearer distinction between terrain that should and should not be traversed by a vehicle. Despite this benefit, many systems still had occasional issues with obstacle false positives (even in static settings). Due to the binary nature of the obstacle/freespace distinction, this would result in traversable routes appearing blocked, and the implementation of sub-optimal low level and high level maneuvers. This form of error is common in systems that only consider obstacles and freespace as opposed to a more analog representation.

The Urban Challenge also provided a context where preferences and tradeoffs between various high and low level behaviors could have a drastic effect on final vehicle performance. Despite the use of different algorithms for local path and trajectory selection, the 6 finishing teams all used some subset of the following considerations when selecting amongst possible actions:

- Time to execute
- Distance traveled
- Distance to static obstacles
- Distance from dynamic obstacles
- Cross-track error from chosen path/lane
- Heading error from chosen path/lane
- Curvature
- Smoothness/change in curvature
- Velocity (sign and magnitude)
- Acceleration

The various implementations of mission planning systems were even more similar, with consideration given to time and distance of a planned route, complexity of encountered intersections and necessary maneuvers, and both the presence and age of perceived blockages. In turn, each team had to weigh the relative tradeoffs related to all of these considerations within their planning systems. In all cases, this was accomplished by manually setting the value of appropriate parameters, and validating the settings through either actual or simulated vehicle behavior. Many teams continued to modify these and other settings during and even after the final qualifying event, in an attempt to correct observed poor or incorrect behavior.

2.2 Mobile Systems in Unstructured Environments

The first full cross country navigation by an autonomous vehicle was achieved with the Autonomous Land Vehicle (ALV) in 1987 [42, 43]. The ALV was an eight wheeled vehicle capable of traversing off road terrain and slopes of up to 15 degrees. It was equipped with an ERIM laser range finder, onboard computing, and a radio link to additional offboard computing. Range data was used to create a local elevation map; this map was then used to identify traversable and non-traversable regions. This identification was made by simulating the effects of the local terrain on the ALV's suspension; patches of terrain that caused a suspension limit, were of sufficient slope, or resulted in insufficient ground clearance were labeled as non-traversable. On board planning was achieved through a reactive behavior based architecture. One behavior was tasked with turning towards a local subgoal and then moving until its path was blocked. Once its path was blocked, the ALV would invoke a second behavior that would consider first gentle turns and eventually point turns to avoid obstacles and face a navigable area. Parameters guiding individual behaviors and their interactions were tuned in simulation.

Subgoals were extracted at key points along long range route plans. These routes were generated offline, using available prior data about an environment, including elevation maps, locations of roads, canopy cover, etc. This information was generally available in rasterized or grid form. It was processed by assigning costs to grid cells based on the available data, and then using A* or other grid planners to generate the minimum cost path to goal [44, 45]. This is a key distinction between the offline and online planning systems of the ALV: the offline route planner considered a continuum of cost values for each terrain patch, while the onboard system only considered a binary notion of traversability. The offline planner's use of costs is an instance of the "weighted region" planning problem [46, 45, 47]. The advantage of such an approach is that it allows a hierarchy of preferences: open areas are better than canopy covered areas, but canopy covered areas are still better than steep slopes. However, this advantage also creates a new challenge: the relative weightings of different terrain features must be determined. The tuning of these preferences in turn has a large effect on the vehicle's eventual long range route plan.

The final route is especially important when it is computed offline and then explicit attempts are made to follow it. Several experiments demonstrated instances where the ALV would diverge significantly from its route due to obstacles, and then struggle to get back onto the route when an easier path to the final goal was available. This lead to work to encode the result of offline planning not just as a final route, but as a route from any point in the environment (encoded in a vector field) [44]. The result of this distinction is a better fusion of online and offline knowledge of the environment. However, taking advantage of this fusion requires a consistency in the form of the data being fused.

It is interesting to note the performance of this early system for the purpose of comparison to modern systems. The 1987 experiments resulted in a system which could repeatedly traverse courses that averaged 600 m in length at average speeds of 3 km/h. The best reported experiment was a 735 m traverse at 3.5 km/h. In the unstructured setting, speed is often a difficult metric to use, as it is highly dependent on the specific terrain and vehicle. For example, in the ALV experiments it is reported that a human driven pickup truck could only traverse the terrain at an average speed of 10 km/h [43].

Following the ALV, a great deal of work took place in the context of the UGV/DEMO II program [48]. An important milestone was the development of stereo vision algorithms that were sufficiently robust and efficient for use in outdoor navigation. This approach was then applied to the Robbie rover, resulting in autonomous traverses on the order of 100m through rugged terrain [49]. This also marked an early instance of stereo vision being used as the sole mode of perception in complex terrain while using solely onboard computing. Obstacles were detected and avoided by thresholding the height of stereo points. Further characterization and modeling of this stereo perception system led to approaches for determining the value of the height threshold and other system parameters to constrain risk, or alternatively to determine the risk of a given parameter configuration [50, 51].

Concurrently, the Navlab II autonomous HMMWV was developed [52]. This vehicle served as a useful testbed for several different components and final systems. Notable was the SMARTY local navigation system and the DAMN system for action arbitration [53, 54]. Using an ERIM LiDAR scanner, Navlab II constructed and fused local elevation maps [55]; obstacles were then defined as regions of high slope or height discontinuities. Possible constant curvature arcs were then scored by the distance from and along an arc to the nearest obstacle. Scores for possible actions were converted to votes, which were then sent to the arbiter. The arbiter would weight the votes of both the SMARTY local navigator, and a goal directed behavior, and choose the action with the highest weighted vote. Additional behaviors could also be added to the system, such as explicit road following or avoiding excessive tilts [56]; however, this required a more careful tuning of the weights between these behaviors.

Different configurations and tunings of this system resulted in different vehicle behavior, capabilities, and results. The system described in [53, 54] contained a goal directed behavior that always preferred to turn directly towards the goal. This system could traverse waypoints on the order of 100m spacings at speeds of 2 m/s; However, the lack of any global planning prevented the system from dealing with cul de sacs and dead ends. In comparison, the system described in [57] used a D* [58] based global navigator that received obstacle information from SMARTY, and could traverse kilometer scale waypoint spacing. The global navigator considered regions as either obstacle, near obstacle, or traversable, and would plan an appropriate policy (treating areas near an obstacle as five times the cost of normal traversable areas); individual actions would then be voted on based on the cost to goal from the end of the arc. A given configuration of the total system could have its behavior adjusted by modifying the relative weighting given to the local and global votes. Another version of Navlab II paired D* with the RANGER [59] local navigator. As opposed to considering just the distance of an arc to nearby obstacles, RANGER simulated the results of each possible control action, resulted in predicted configurations of the vehicle for each potential action. As with the ALV, constraints on tilt, ground clearance, and stability were enforced to ensure vehicle safety; the safe action preferred by the global navigator was then executed. This configuration eventually resulted in reliable mobility over distances of 15km at average speeds of 7 km/h [60].

The UGV/DEMO II program produced 4 key demonstration events. Demo A in July of 1993 used tele-operated vehicles. Demo B in June of 1994 demonstrated semi-autonomous navigation over several hundred meters. Demo C in July of 1995 featured multiple autonomous vehicles navigating cross country terrain. The final Demo II was held in May of 1996, and featured autonomous traverses of multiple kilometers [61]. The success of the DEMO II program led to the creation of the Demo III program [62]. In a similar manner to DEMO II, this program resulted in not only integrated systems on the target eXperimental Unmanned Vehicle (XUV) platform, but additional prototype systems that allowed for easier component research and development. Additionally, one of the technology goals of DEMO III was to create mobile systems that could factor in tactical considerations (such a stealthiness) when determining local and global actions, instead of purely mobility based considerations.

Building on this goal, development of the NIST autonomous HMMWV [63] led to a perception system that separated terrain into three classes instead of two: ground (traversable), obstacle (non-traversable) and canopy/overhanging [64, 65]. This third class represented traversable terrain that would be stealthier than open ground. By modifying the relative costs or preferences with respect to covered versus non-covered terrain, the vehicle could be tuned to exhibit different degrees of stealthy behavior [66]. Different sets of actions through equally traversable terrain were also preferred based on the necessary changes in steering angles [63]. Additionally, speed was an important consideration as the system was capable of speeds up to 35 km/h. This led to the introduction of parameters to increase the berth given to obstacle at higher speeds [67].

Component technologies from the NIST system were also ported to the XUV platform. This implementation [68] made use of the planning sub-system described in [63], which considered a precomputed set of possible actions based on possible changes in speed and steering rate. As opposed to simply classifying terrain as traversable versus non traversable, the perception system produced an analog estimate of the density or porousness of obstacles [69]. A cost was assigned to patches of terrain based on these densities and other terrain features, such as roughness, predicted ground clearance, and the degree to which the vehicle had directly observed the terrain, as well as more high level properties such as stealthiness [70]. The model mapping from these features to costs was implemented as a linear function with a set of weight parameters controlling the relative contributions of different features. A rule-based model was also used to determine preferences based on higher level considerations, such as obeying traffic laws [71]. By using different parameters and rules in various preference models, different overall behaviors were demonstrated such as preferring roads, attempting to stay hidden, or even attempting to stay visible to a stationary supervisor.

Research utilizing the XUV platform continued under the Robotics Collaborative Technology Alliance (RCTA) with a great deal of full system integration and testing, as well as additional work on component perception technologies. For instance work at JPL [72, 73] continued to explore more complex notions of traversability. Patches of terrain were first identified as potential positive or negative obstacles based on their geometric properties. Positive obstacles were then classified into one of a set of material classes (e.g. soil, rock, green vegetation, dead vegetation, etc.) based on their perceived color. A rule-based model then mapped from geometric properties and the material class to a traversable/non-traversable distinction. Later work also explored this classification approach using 2-D LiDAR statistics [74] and image texture [75]. Along with more intelligent geometric approaches to obstacle detection [76], these approaches were later fused into a single, robust obstacle classification scheme [77].

Additional work involving terrain classification from richer data sets was also performed on the XUV. Advances in sensing and positioning accuracy, along with increased computational power allowed for creation and classification of full 3D point clouds from LiDAR, as opposed to approaches that processed 2D line scans or 2.5D elevation maps. As with previous classification approaches, statistical properties of the input 3D data were computed and used to classify terrain into different semantic categories [78, 79]. Additional work allowed for a hierarchy of classifiers at the data fusion level, in an attempt to reduce the necessary human parameter tuning [80]. More recent work has also explored the use of more contextual information to aid in classification of adjacent areas [79, 81].

Given how important accurate perception and terrain understanding had proved to be to autonomous navigation, the PerceptOR program was undertaken with the goal of "understanding this critical enabling perception technology as it applies to robotic mobility" [82]. Much of the work performed under the PerceptOR umbrella resulted in specialized approaches for detecting vegetation [83, 84], water [85], trees [86], and negative obstacles [87, 88]. Further research into stereo vision for mobile platforms was also performed [89, 90, 91]. In addition, several fully integrated systems were developed to further evaluate and validate these technologies for autonomous navigation [92, 93, 94, 95]. One of the core principles of this effort was to perform unrehearsed experiments; the various mobility systems would be evaluated on terrain on which there had been no prior testing, challenging the systems to be both both generalizable and robust [96]. One of the results of this evaluation approach was that individual system tuning and testing was given more focus, and had to be performed over a much wider range of potential environments and operating scenarios.

Another characteristic of the PerceptOR program was its use of prior and remotely collected environmental data. Previous uses of prior data [45] for mobile robot navigation had dealt with processed, GIS style data sets such as road networks, elevation maps, and canopy maps. However, PerceptOR (and to a degree DEMO III) made use of unprocessed prior data such as aerial or satellite imagery and aerial LiDAR scans. In this way, processing prior data became yet another perception task, using similar techniques and requiring similar solutions [97, 98, 99]. Similarly, PerceptOR also at times made use of a separate robotic helicopter, that could be tasked with autonomously observing terrain prior to the ground vehicle encountering it. Although the use of prior or remotely collected data would seem to be a clear performance enhancer, it also provided new challenges. The use of multiple heterogeneous sources of perceptual data creates an interesting data fusion problem. Fusing the data early in a processing pipeline leaves open the possibility of data not being available in all locations for all sensors; a separate processing function for each possible subset is therefore required. However, fusing the data later in the pipeline requires the results of each individual sensor processing component to "speak the same language"; that is, the relationship between the units of each result must be known or derived. When these individual results are a more abstract quantity such as cost, it adds to the difficulty of designing the individual processing functions [95].

Overall, results from PerceptOR were highly terrain dependent. Average speeds for a test site ranged from 0.15 to 0.65 m/s , and were on average approximately a third of the speed with which the system could be tele-operated. Over the life of the program, there were approximately 130 km of official autonomous traverse. During this time, the average number of safety interventions per kilometer traversed decreased by a factor of 20 [96]. Use of prior data was not found to significantly affect mobility performance in most instances.

Serving as a follow on to the PerceptOR program, the UPI program combined the use of a custom designed robotic mobility platform (Crusher) [100] with an autonomy system descended from PerceptOR [101, 102]. As with PerceptOR, UPI continued to leverage machine learning approaches for better terrain understanding. Additionally, online adaptive learning was also demonstrated [103, 104]; as with the Grand Challenge, this served as further validation that online adaptation could be a part of a reliable system. The UPI program also made further use of prior terrain data; however, in contrast to PerceptOR prior data was shown to have a significant positive affect on vehicle performance[2] [105]. Over the length of the program, Crusher autonomously for several thousand kilometers, including over a thousand kilometers of experiments in unrehearsed complex unstructured terrain. Average speeds were highly dependant on the difficulty of the terrain and the amount of prior data available; at times the average speed approached 14 km/h. This was sufficiently fast over complex terrain that safety teams in pickup-trucks or HMMWVs often had difficulty keeping up with Crusher, occasionally incurring damage to the chase vehicles in an effort to stay close.

Occurring in parallel to the Demo III and PerceptOR programs was research and development geared for the use of robotics for scientific and planetary exploration (as opposed to specifically terrestrial navigation). A planetary mobile robot has one distinct advantage over terrestrial systems, in that there is no vegetation; the supporting ground surface is usually directly observable. However, this advantage is more than negated by the complex nature of the ground surface, often requiring precise understanding of the local terramechanics. Additionally, as opposed to motion planning purely for mobility purposes, there is also the need for higher level scientific mission planning, and the decomposition of such high level goals into lower level navigation goals [106]. For instance, planetary mobile robots must no only choose where and how to drive based on what is traversable, but on the energy expenditure [107, 108]. Similarly, instead of simply striving to traverse to a goal region, a scientific robot must also determine what to study and when, balancing the relative tradeoffs between difficulty of traverse and scientific value [109, 110, 111, 112, 113]. Even relative preferences amongst different scientific targets must be considered.

Of course, with planetary robotics the primary challenge to date is not science autonomy, but that of designing, building, transporting, and operating a reliable autonomous mobile system that just happens to be on another planet. While previous planetary rovers had been completely teleoperated in 1997 the Sojourner rover provided the first demonstration of rover autonomy in space [114, 115, 116]. Utilizing low resolution stereo range data, Sojourner was capable of low level reactive autonomy to prevent collisions with obstacles, as well as specific pre-programmed behaviors that could be commanded remotely. In 2004, the MER rovers Spirit and Opportunity began autonomous operation and exploration on the Martian surface. As with Sojourner, MER utilizes stereo vision for terrain assessment and (additionally) position estimation [117]. The rovers can operate with varying degrees of autonomy and safe-guarded tele-operation [118, 119]; the tradeoff is often that of substituting human planning time for autonomous planning time. When operating in its most autonomous mode, the rovers utilize a terrain assessment approach named GESTALT [120, 121, 122], derived from an earlier approach named MORPHIN [123, 124, 125]. These approaches are geometry based, and assign continuous cost values based on the slope and roughness of various terrain patches. Possible local motion arcs are then scored according to these costs, as well as the difficulty of achieving a motion and progress towards a goal. As with previous arbiter based schemes, the final action is chosen through a weighted vote. A global planning vote was added to the rover's software in 2006 [126, 127]. To date, the two rovers combined have driven more than 25 km across the Martian surface demonstrating high reliability for a mobile robotic system under extreme conditions.

[2] It is the opinion of the author that this difference may have been primarily due to increased waypoint spacing, which places more of burden on autonomous route planning

2.3 Recent Trends in Mobile Robotic Systems

From this brief look at the evolution of mobile robotic systems over the past several decades, a number of long term trends and milestones can be identified:

- Geometric perception techniques have greatly increased in accuracy, resolution and complexity, along with the availability of associated sensors.
- Appearance and semantic based perception has become integral (both in semi-structured and unstructured environments) for identifying structures from road surfaces and signage to vegetation to different surface materials. Making these distinctions accurately has in turn become more critical to reliable and robust autonomy.
- Perception systems have increased their use of machine learning techniques, perhaps most usefully in the use of supervised semantic or material classification.
- As opposed to simply modeling a (presumed) static world, perception systems have become more and more capable of identifying, tracking and predicting the future location of moving obstacles.
- It is becoming less and less common for perception systems to only produce a binary distinction of the traversability of terrain. Instead perception is increasingly required to produce a higher dimensional description of terrain, and a more continuous measure of its preferability.
- Planning systems for autonomous navigation have grown from simple location based path planners and reactive controllers to hierarchical motion planning systems that can operate in kinodynamic state spaces. Arbiters are becoming less popular for determining low level controls; instead the recent trend is for low level planners to use high level planners as guidance.
- Planning systems have made more extensive use of predictions or simulations about how the world will evolve and interact with the robot, from predicting where moving objects are likely to be in the future to simulating how the robot will respond to certain terrain features when making specific maneuvers.
- As opposed to only trying to identify or produce a feasible motion plan, planning systems are capable of performing multi-criterion optimizations over such factors as maximizing velocity, minimizing accelerations, obeying rules of the road, etc.

A unifying thread amongst many of these trends is that a robot's model of the world (produced by perception) has grown in complexity and dimensionality, while a robot's planning system is now tasked with producing longer range plans over state and actions spaces of similarly increased dimensionality. In addition, along with more complex world models and planning states, there are now a range of considerations that must be accounted for, and it is no longer concretely defined what is the correct, desired, or optimal behavior for the robot. This last issue proves to be of critical importance, and will be explored in detail in the next chapter.

Chapter 3

Behavioral Preference Models in Mobile Robotic Systems

Comparing the systems described in the previous chapter to the requirements described in Section 1.3 reveals much that remains lacking

Reliable Some autonomous mobile systems have achieved high levels of reliability, notably the Grand and Urban Challenge finishers, and the MER rovers. There is a clear correlation between the amount of validation testing of the final (frozen) system, and the reliability of said system. Unfortunately (as lamented in [36]) such testing is currently the only way to properly ensure reliability. Reliability also appears correlated with the complexity of the operating environment; systems operating in unstructured terrains have higher rates of component errors that lead to system failure.

Robust Achieving robustness requires proper interaction and layering of multiple components and sub-systems, to account for potential component failures or errors. It can also require making decisions or choosing actions that would not normally be considered. For example, multiple Urban Challenge teams reported instances in either the qualifier or final competition where their vehicle become extremely close to either a real or perceived static obstacle. In some cases very tight wiggle maneuvers resulted in sufficient clearance to continue normally, while in other cases only pose drift allowed the vehicle to move again. While it may seem prudent to permanently eliminate certain sets of actions (such as those that move too close to perceived obstacles), there may be novel scenarios where all actions must be at least considered for robustness.

Rapid Development For the most part, rapid development and deployment still escapes the mobile robotics community, as most capable systems require design and development times on the order of at least a year. Some of this is to be expected, as robotics remains an emerging technology, and research projects by their nature are attempting something novel. However, a great deal of time is still spent tuning and then validating system performance[1]. At the moment, reliability and rapid development are in direct conflict and seem almost mutually exclusive.

[1] As an example, [95] reports "integration and tuning on the PerceptOR program alone required perhaps 30 man years of effort"

Reusable Much of the time spent on system development goes into tuning a system for a specific environment or scenario, rather than continuing development of component capabilities. Therefore, the results of this effort provide little benefit towards development of future systems, and disappear once the current system is no longer used. Further, a great deal of this effort must be repeated if the system or a component design is changed significantly, or the environment or expected use scenario is altered.

While great strides have been made in the last 20 years towards advancing the capabilities of mobile systems, there clearly remain several barriers to achieving the above requirements. One such issue is the complexity of the tasks mobile systems now deal with. On the ALV, the perception system was tasked with simply making a binary traversable or non-traversable distinction, and the planning system simply chose actions that did not result in crossing untraversable terrain and drove closer to a short-range goal. In contrast, the XUV or Crusher must consider a range of traversability measures, and the Urban Challenge scenario required considerations of safety, time, distance, and rules of the road.

The end result is that as the complexity of both the systems and their tasks increase, mobile robots are increasingly producing and considering continuous preferences over both where and how to drive. As internal models of both the environment and potential vehicle interactions become more accurate and informative, it becomes increasingly difficult to identify the proper behavior, even when only considering traversability. However, additional considerations are also being added to the equation, such as monitoring energy usage [107, 108], remaining stealthy [71, 128], or performing a socially acceptable driving maneuver or behavior [129].

Unfortunately, the issue does not stop at just identifying the proper behavior given the context. If mobile systems are going to be truly robust to the unexpected and reusable in new environments, they must be able to choose the correct (or at a minimum reasonable) behavior in contexts for which they have never previously been explicitly tested. This requires that mobile systems be encoded not with the correct behavior, but with the mechanisms for choosing the correct behavior given the context; a model of preferences over various behaviors and choices is necessary. As world models, simulated interactions, and planning considerations continue to increase in sophistication, interpreting this complexity into preference models[2] over possible decisions becomes increasingly challenging.

3.1 Continuous Preferences over Behaviors

The issue of continuous preferences has perhaps been most evident in the perception domain. Systems that used a binary measure of traversability have clear limitations. In unstructured environments, this results in no distinction between seemingly traversable but distinctly different terrains (e.g. there is no way to encode that one ditch crossing site is safer than another if they both meet a traversability threshold). The result is that binary measures of traversability produce either overly aggressive or conservative behavior (Figure 3.1). Similar problems can occur in semi-structured environments (see Figure 6.1), as demonstrated by Urban Challenge systems that preferred to

[2] In this document, a preference model refers to a mapping from a state or decision to a scalar cost, reward or utility value. This is in contrast to other work [130] which uses a probabilistic mapping representing the likelihood of a desire to move between semantically defined states.

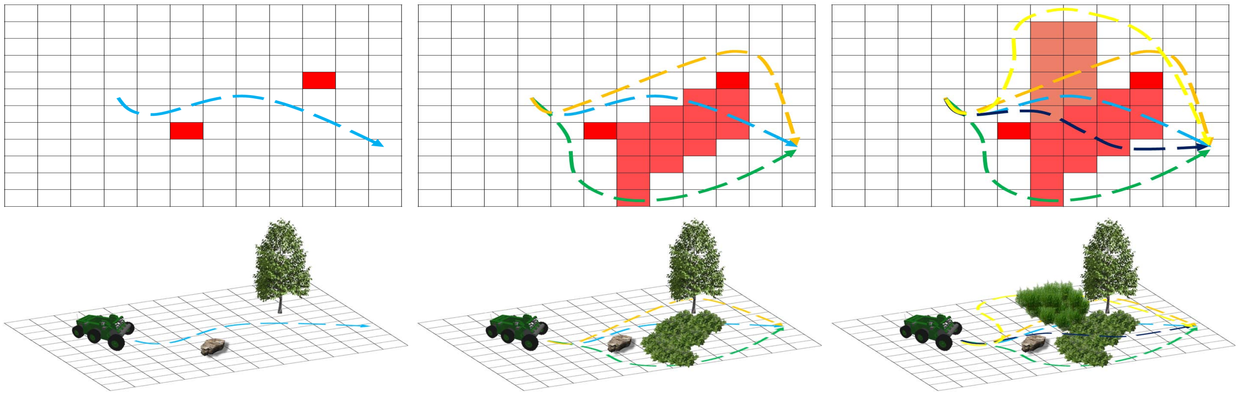


Figure 3.1: Examples illustrating the effect of different preference models on a robot's behavior. **Left:** In simple environments with only well defined obstacles and freespace, a wide range of preferences will produce nearly equivalent behavior. **Center:** With a single class of intermediate terrain (bushes), several different paths are optimal depending on the relative preference between bushes and freespace. **Right:** With multiple classes of intermediate terrain, the variety of paths that could be optimal (depending on the relative preferences between bushes, grass, and freespace) is further increased. In such scenarios, the tuning of a preference model will dramatically affect a robot's behavior. If only binary traversability were utilized, the variety of possible behavior would be diminished.

remain permanently stationary than risk a incredibly low speed collision. Human drivers occasionally encounter similar scenarios (e.g. tight parallel parking), and understand that actions that may result in very low speed collisions, while undesirable, are preferable to never moving. As these issues have become better understood, the result has been a move to systems that use a more continuous measure of traversability, such as [68, 92, 95, 121, 101, 102, 124, 125].

While using an analog definition of traversability allows more intelligent behavior, it also increases the complexity of deploying a properly functioning system. Systems that used a traversable or non-traversable distinction only had to solve a binary classification problem. This mapping to traversable or non-traversable completely determined the behavior of the robot (with respect to where it preferred to drive); essentially, the desired behavior of the robot was encoded in the classification function mapping perceptual data to traversable or non-traversable. This mapping often took the form of a small set of thresholds on various terrain properties (e.g slope, step height, etc.).

However, in a system with an analog measure of traversability, in essence a full regression problem must be solved. That is, the desired behavior of a robot is encoded not in a mapping from terrain properties to a binary class, but in a model mapping from terrain properties to a scalar value that encodes preferences. When the desired behavior is to maximize traversability or safety, this measure is often called traversability cost or mobility risk, or just cost for short. This mapping from terrain properties to cost (the cost function) is far more complex than a binary mapping, as it encodes far more complex behavior (through continuous output). Therefore, as the behaviors and actions that mobile systems are expected to exhibit become more complex, the task of encoding these behaviors in a generalizable model of preferences will become both more difficult and more central. Unfortunately, this problem has not received a great deal of focus; it is often only briefly mentioned in the literature. With respect to costs defined over patches of terrain, the mapping to costs from terrain parameters or features is rarely described in detail, usually with the caveat that it was simply constructed to provide good empirical performance.

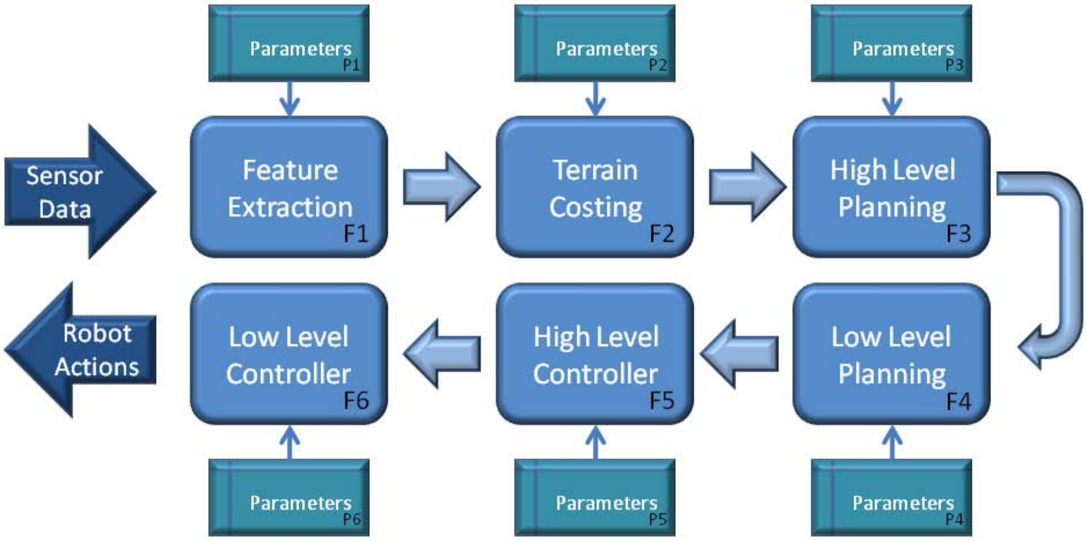
For instance, the MORPHIN [123, 124, 125] and GESTALT [120, 121, 122] systems simply assume direct linear relationships between sets of terrain features and traversability, with user-defined thresholds denoting maximum cost obstacles. A similar approach is also applied in [131, 132, 133], where cost is a weighted combination of geometric properties such as terrain slope and roughness. The NIST system for DEMO III [70] also assumed a linear relationship for both local and high level terrain features, with user tuned relative weightings of different features. Rule-based [71, 86, 106] or fuzzy logic [134, 135, 136, 137, 138, 139, 140, 141] approaches also require explicit human definition of the rules for of costing. [142] uses a cost function proportional to the probability that the terrain is traversable; however, this probability is based solely on a simplistic sensor model, and therefore is essentially a smoothed version of binary traversability. Further, it requires a user-defined weighting between probability and distance. Other work [57, 143, 144, 145, 146, 147] simply assigns different costs to different terrain patches or classes with little mention of the intuition behind these decisions, or how the mapping is implemented. The current state of defining preferences over terrains is perhaps best summarized in [95][3].

Preference models are also necessary in the planning domain, and are often similarly constructed. For instance, systems that utilize arbiters [52, 53, 54, 56, 57, 92, 123, 124, 125, 136, 138, 126, 127] often provide little intuition for relative weighting between votes from different behaviors or modules, aside from empirical validation. More deliberative planning systems are generally tuned in a similar manner to preferences over terrain patches, by manually modifying preferences over prospective actions until desired behavior is achieved [36, 37, 38, 39, 40, 41, 63, 71, 148]. Higher level route or mission planners must also make use of preference models in order to balance metrics such as time, distance, risk, energy, observability, etc. This problem is solved one of two ways. In some instances, there is only a desire to constrain some metrics while optimizing another [45, 107, 108, 149, 150]; this still requires an accurate model of the metric to be optimized. In the more complex case, there are no hard constraints, and the desired behavior is a weighting between various models (i.e. a multi-criterion optimization) [66, 70, 71, 109, 110, 111, 112, 113, 129]; the source of this weighting is again often manual tuning.

3.2 Manually Constructing Behavioral Preference Models

The result of this purely empirical approach is that most mobile systems are simply manually ’tuned’. That is, designer intuition is used to construct a parameterized model mapping from terrain or action features to scalar preferences or costs. A final set of parameter values for the model is then determined through what is essentially a generate and test procedure. Candidate parameter values are produced by an engineer, and the system is then tested to see if it performs the correct (or at least acceptable) behavior. As long as the system does not behave in a desirable manner, the model and its parameters are continually modified and each new configuration tested

[3] “The costs used represent (roughly) mobility risk. The tuning of these costs/risks emerged as a major design issue during the execution of the work. Correct system behavior was only obtained by squarely addressing the question of what amount of extra excursion is justifiable to avoid a given level of risk. Nonetheless, we can report few principled methods beyond understanding that the tradeoff exists and tuning the costs in order to get correct behavior under various circumstances.” [95]



$$A = F_6(P_6, F_5(P_5, F_4(P_4, F_3(P_3, F_2(P_2, F_1(P_1, S))))))$$
$$\frac{\partial A}{\partial P_i} = (\prod_{j=n}^{i+1}[\frac{\partial F_j(P_j, \centerdot)}{\partial F_{j-1}(P_{j-1}, \centerdot)}])\frac{\partial F_i(\centerdot, F_{i-1})}{\partial P_i}$$

Figure 3.2: A sample mobile robot architecture, represented in a feed-forward manner. Application of the chain rule demonstrates how the effect of changing a parameter on the final output can be difficult to determine the earlier the parameter is used in the system.

until the engineer is satisfied. Testing a particular model can take many forms. It can involve simply visualizing the resulting output of a specific configuration[4], or it may involve full system testing.

Unfortunately, such a manual approach is rife with problems:

- It requires a good amount of intuition as to how the modification of particular parameters will affect the end behavior of the system. Without such an intuition, the engineer will essentially be performing a local 'hill-climbing' style optimization. Since each parameter presents a trinary decision (increase, decrease, or remain constant), there are 3^D possible unit changes for a D dimensional parameter space.

- Even for an engineer well versed in the system design, this intuition can be hard to come by. This is especially true the more removed a parameter is from the final behavior. If a robotic system is viewed not as a continues loop but as a feed-forward network (mapping sensor inputs and perceived state to behaviors), then each parameter's effect on end behavior can be computed via backpropagation [151]. As with backpropagation in neural networks, the more layers between a parameter and the final output, the more the error signal degrades due to repeated application of the chain rule (Figure 3.2). This lack of intuition only gets worse as the dimensionality of the parameter space is increased.

[4] [125] specifically mentions the construction of visualization tools for this purpose

- With a properly designed preference model, good intuition or a sufficiently low dimensional parameter space, manual optimization can sometimes produce an approximately tuned result quickly. However, there is a rapidly diminishing return on the investment of human resources. Once an approximate tuning is achieved, it is even more difficult to determine the effect of a parameter change on the output behavior. Additionally, one never knows when the true optimal result has been achieved. Therefore, if the system later demonstrates poor performance, it is unclear whether the preference models are simply not appropriate, or if the problem lays in system design or component capability.

- The system must be tuned not just for one scenario but for a sufficiently representative sample. Each evaluation of a new setting must consider all such scenarios, otherwise a setting that improves performance in one instance may hurt performance in others. Therefore, any parameter modification requires validation through a large set of regression tests. If these tests must be performed at the full system level, they can be enormously time consuming.

- It may be necessary to design and tune multiple models for the same task, when the system inputs are not constant (such as in the case of fusing multiple sensors). It may also be necessary to tune these models such that the result outputs are compatible. For instance, when costs generated independently from different data sources are fused [95, 101, 102], this adds an additional constraint over all cost functions: each function must reproduce the correct behavior both individually, and when fused. Essentially, costs from different sources must all be in the same ’units’.

- Even if all of the above issues are overcome, it may be necessary to repeat this process of model design and hand parameter tuning whenever
  - The system design or data flow is changed
  - A component design or implementation is modified
  - A significant failure is observed that can be traced to parameter tuning
  - The desired behavior or operating environment of the system is significantly changed

The issues associated with determining preference models over behaviors does not only result in potentially large amounts parameter tuning; it can also affect system design. Specifically, component or sub-system solutions that result in a high dimensional interpretation problem are often simplified, condensed or avoided, even if the resulting solution is less powerful or generalizable. The result is systems that essentially throw away or ignore potentially useful information, because hand tuning a mapping to interpret this extra data into behavioral preferences is too complex (see Section 4.1 for examples).

3.3 Problem Statement

The advancement of mobile robotics has resulted in the production of models of environments and potential plans of action of ever increasing complexity. At the same time, the various considerations that a mobile robot must take into account have greatly expanded from simply ensuring robot safety and task completion. These trends have resulted in the increasing difficulty of a robot’s task of choosing its next behavior, and have led to approaches that require a clear ordering of preferences over potential actions to take and terrains to traverse.

Current approaches to producing preference models over possible behaviors are a clear barrier to the previously defined requirements for mobile systems. With these manual approaches, reliability and rapid development are in direct conflict: reliability can only be achieved through a tedious and time consuming process of manually constructing, tuning and validating the necessary models and parameterizations. Additionally, manual approaches tend both to limit the scope of scenarios for which the system is tested, and to overfit to those scenarios; this serves to hinder the robustness and reusability of mobile systems when they encounter novel scenarios. The possibility of having to repeat this process if the system or its operating considerations change is always present. Finally, it is never fully understood how well the preference model has been optimized, making it difficult to understand which parts of the system require improvement when presented with poor system performance.

As opposed to manual design and tuning, automated approaches offer an attractive solution. An automated process that required significantly less human interaction but produced similar results would seemingly allow for both reliability and rapid development to coexist. Additionally, an automated approach would have several advantages that might allow for better performance. Even with rapidly diminishing returns, an automated process could continue to improve the model and its parameterization if the only cost is computational cycles. Further, an automated process would have the advantage of being able to consider numerous scenarios at once, and would be less likely to continually contradict itself. Finally, an automated process could handle much higher dimensional parameter spaces than a human engineer reasonably could be expected to.

This last advantage might even impact mobile robot system design. If the complexity and dimensionality of world models, simulated interactions, etc. is no longer a concern, then there is no longer an incentive to simplify these constructs purely for ease of implementation[5]. Further, a large portion of the human resources and testing time that are currently devoted to hand parameter tuning and validation could instead be used for further system development.

Therefore, the problem statement of this thesis is as follows: ***The performance of mobile robotic systems in complex environments and scenarios is increasingly dependant on the proper design and parameterization of preference or utility models that are difficult and costly to construct and tune manually. Principled, automated methods of determining and validating preference model design and parameter settings are required that involve at least an order of magnitude less human interaction, while producing equivalent if not better system performance.***

As this problem statement mentions parameter tuning, a final clarification is necessary with regards to parameters and the scope of this thesis. Parameters in a mobile robotic system fall into one of the following general categories:

- Physical parameters that are directly measurable (e.g. vehicle dimensions)

- Model parameters that are directly measurable or can be empirically observed (e.g. center of gravity)

[5] This is not to imply that more complex representations are always better. Rather, the complexity of a representation should be based on how well the representation is correlated with that which it is attempting to model, not on how difficult it is for a human to interpret the representation

- Model parameters that can be computed analytically or through simulation (e.g tip-over angles)
- Control parameters that can be computed analytically (e.g. PID gains)
- Rate or resolution parameters that involve tradeoffs in computational resources (e.g. internal map resolution)
- Environmental parameters that can be measured for a specific terrain (e.g. friction coefficients)
- Unit-less parameters that help define relative costs, benefits and preferences (e.g. cost function parameters), and have a direct or indirect effect on robot behavior

Physical and model parameters are generally considered as constants of a specific vehicle. Control parameters require proper tuning, but the formalism of controls theory provides the necessary framework. Rate and resolution parameters require design tradeoffs only if there is a scarcity of resources; in this case a system cost-benefit analysis is usually performed, often with the aid of experimental performance data. Environmental parameters can be measured or learned once if operating conditions are not expected to change drastically; if this is not the case then the framework of self-supervised learning provides a clear solution (Section 4.2).

Therefore, this thesis addresses only the final category of parameters: those that help define a model mapping from world or action models to preferences over various actions and terrains. Further, the problem is not only to determine the proper parameterization of a specific model; it may include constructing the model from scratch.

Chapter 4

Automating Construction of Preference Models

Many previous attempts at principled approaches to defining preferences over behaviors have resulted in manually designed and tuned solutions. These solutions are often based on sets of thresholds or simple functions, and are often designed to perform well at the extremes. That is, states or actions that are close to ideal or to be avoided at all costs are usually ranked as such. The result is systems that can perform well when only faced with these extremes (such as when operating in structured environments). However, it is when environments and considerations become more complex, with no obvious 'right or wrong' choices, that these approaches begin to break down. Since it is not only necessary to order possible behaviors, but to assign accurate costs to continually weigh and consider possible behaviors , the hand tuning task becomes more and more time consuming. As the complexity of possible behaviors increases, this problem approaches intractability: for example, weighing the relative cost of traversing two patches of terrain is hard enough, but what if traversing one patch involves a much harder turn than the other as well? Attempting to hand define every possible preference over such decisions is clearly an exercise in futility.

Attempts to formalize the manual construction of preference models still result in a parameter tuning problem. For example, the MORPHIN [123, 124, 125] and GESTALT [120, 121, 122] approaches define reusable preference models over terrains; however, each model must be properly parameterized in order to result in a robot achieving the proper behavior. The same is true with rule based or fuzzy logic [134, 135, 136, 137, 138, 139, 140, 141, 132] approaches to traversability. On the planning side, arbiter based systems provide a formalism for the weighing of different behaviors with different individual goals, but again the proper vote weighting must be hand defined to produce desired behavior.

Another approach to engineering this problem away is the use of accurate physical simulation, such as the previously described RANGER system [59] or the ALV [42, 43]. More recent work in this area can be found in [152, 153, 154, 155, 156]. However, simulation does not solve the problem; at times it can make it even harder[1]. Using an accurate simulation to predict the consequences of a specific behavior (in terms of its affect on the kinematic or dynamic state of the vehicle) necessitates a preference model defined over these consequences. Again, for certain structured environments this may be easy; known vehicle tolerances and thresholds can define

[1] If the simulation results are higher dimensional than the alternate feature space

kinematic or dynamic states that are certain to cause failure; However, in unstructured environments the task remains to construct a model mapping from a high dimensional description of a behavior to preferences over possible behaviors.

It would therefore appear that more automated approaches to constructing preference models can not be based on simply engineering the problem away. As opposed to an intensive manual effort solely to construct and tune a preference model, this work considers a different approach: rather than *hand tune* a model and its parameterization, *learn* a model and its parameterization. Machine learning techniques have continually proven themselves to be first useful and then essential to producing high performance mobile robotic systems. Therefore, as opposed to manual tuning, it is proposed that mobile robotic systems be trained to produce the desired behavior.

Some applications of learning for mobile robotic systems have taken the approach of end-to-end learning [157, 158]. That is, they attempt to completely replace all mid and high level software by learning to map directly from raw proprioceptive and exteroceptive inputs to control signals. This approach results in the construction of a single complex learning task. In contrast, this work takes a different approach. Enormous progress has been made in the realms of mobile robot perception and planning; there is no reason not to reuse that effort. Therefore, individual component problems for which manually designed and engineered approaches have proven effective will continue to utilize these solutions; learning will be applied where engineering efforts have proven ineffective or inefficient. Essentially, both human engineers and machine learning algorithms will solve the problems for which they are best suited.

The remainder of this chapter explores previous work in machine learning as applied (directly or indirectly) to the task of learning preferences over possible mobile robot behaviors.

4.1 Expert Supervised Learning and Classification

Supervised classification is perhaps the machine learning technique most commonly used by mobile robots, especially in the realm of perception. The intuition behind its use is straightforward: while an expert may have difficulty in designing rules to classify different patches of terrain, he can much more easily define the class that each patch should belong to[2]. Rather than manually constructing rules to map from perceptual features to classifications, a learning system can automatically generate the necessary rules or models, given examples of sets of features and their desired classification.

The primary advantage (with respect to autonomous behavior) of performing a supervised classification is a remapping of a perceptual feature space into one that is lower dimensional and potentially more intuitive. Some perceptual features have an intuitive, monotonic relationship to metrics such as safety and speed. For example, the rougher a patch of terrain, the more dangerous it is to traverse; this intuition allows for easier hand tuning of parameters that depend on roughness. However, many perceptual features do not provide this intuition; for example, if a terrain patch has a high blue tint, does that make it more or less dangerous?

If a supervised classification stage is inserted after perceptual feature generation but before costs are produced from said features, it can enormously simplify the parameter tuning problem. The dimensionality of the problem is significantly reduced, and each feature has a clear intuition

[2] if the correct classification is not known by the expert, than the problem is ill-posed, and there would be no possibility of manually constructing a solution

behind it, especially if the classes are defined as semantic or material distinctions (bush, rock, tree grass, dirt, etc.). For this reason, this approach has been widely popular for the purpose of perceptual interpretation in unstructured environments, and has been used in the context of the ALV [159], DEMO II/III [160, 74, 75, 76, 77, 78, 79], PerceptOR [83, 84, 95] and UPI [101, 102] programs, in addition to numerous other mobile robot contexts [140, 80, 117, 161, 162, 163, 164, 165]. It has also been widely used for prior data interpretation [97, 98, 99, 105, 166] and terrain classification from proprioception [167, 168, 169, 170].

However, while supervised multi-class classification certainly makes hand tuning a cost function more tractable, it does not actually solve core the problem; the parameter tuning task has only been simplified. While classifier outputs may have a more intuitive relationship to the correct behavior, it can still be difficult and time consuming to determine the proper relative weightings of various classes. Additionally, there is no guarantee that the selected taxonomy is relevant to mobility and behavior. It may require considerable effort for a classification system to distinguish varying classes of vegetation, but that effort is wasted if those vegetation classes are indistinguishable with regards to mobility. Likewise, it may be important to distinguish between types of rocks with near identical appearances, if one class tends to be less weathered and rougher. Defining classes specifically with regards to relative mobility [171, 172, 173] somewhat works around this problem, but makes data labeling far less intuitive. Finally, while the classification approach may make it easier to define a cost function with respect to patches of terrain, the challenge of balancing those costs with costs over actions remains unchanged.

Even if supervised classification eases the task of tuning a final terrain preference model, it can have additional negative consequences if leaned on too heavily. One issue is the potential loss of information. If classification is used to compress a high dimensional perceptual space into a lower one, it can be viewed as a form of dimensionality reduction, with the axes defined by the classes. However, as stated previously, such classes are often defined based on semantic or material properties, not mobility properties or other behavioral considerations. Therefore, this dimensionality reduction may not take into account the final use of the data, and can potentially obscure or eliminate potentially useful information (while at the same time potentially emphasizing potentially useless information). Furthermore, classification is often a very non-smooth mapping, which can further exacerbate the problem (small errors in classification can produce large errors in the resulting cost, and vice versa). Finally, while supervised classification can reduce the time and effort required in one tuning problem, the total effort throughout the system is not necessarily reduced; the classification system now must be trained. Labeling a large representative data set for training and validation also entails a significant manual effort. Additionally, whenever the classifier changes, due to perception system change or additions to the training set, the costing of the classifications must be re-examined and possibly re-tuned.

An alternate approach that makes use of supervised classification has been to treat the task as a specific two class problem: classifying terrain as either traversable or non-traversable. Such an approach could result in either a binary output of traversable or non-traversable (as used in older systems without continuous terrain preferences), or a continuous probability of belonging to the traversable class. [174] makes use of supervised labeling of both traversable and non-traversable terrains. As opposed to offline labeling, [30, 175, 176] produce examples of traversable terrain by observing what terrain a human drives through[3]; nearby terrain that is not traversed is treated

[3] This approach is fundamentally different from standard imitation learning in that demonstration is used purely as

as having been labeled non-traversable/non-preferable in a noisy manner. [177] uses a similar data collection approach, but only for labeling of traversable terrain; explicit examples of non-traversable regions are not required. [137, 138] use human labeling of a fuzzy traversability value as opposed to a hard classification; these labels are then used to tune parameters within a previously designed traversability model.

The fundamental issue with this approach is the assumption that all that matters is whether terrain is traversable or not. It does not address the issue of relative preferences of terrain. For instance, a small rock and a small bush may both have an equal medium probability of being traversable; however, one could argue that a robot should still heavily prefer the bush to the rock to minimize potential damage (or the rock to maximize traction). At the other extreme, a road and an open field may both have a high probability of traversability, but the road is most likely preferable [4]. Additionally, traversability is rarely the only metric under consideration, even for conservative systems. For example, open road may appear to have a 100% chance of being traversable, but that does not imply that driving on a road is free of risk; the longer a robot travels over any terrain, the likelihood slowly increases of a system failure of some form. Therefore, traversable probability and distance must be relatively weighed. The end result is that an analog probability of non-traversability does not easily map to a preference or cost, and (as with multi-class approaches) this mapping must be re-tuned whenever the system is changed anywhere along the input pipeline.

4.2 Self-Supervised Learning from Experience

In contrast to learning approaches that require explicit supervision from an (usually human) expert, self-supervised approaches require no expert interaction. Instead, a robot uses its own interactions with the world to slowly learn how to interpret what it perceives; essentially the robot is learning and adapting from experience. The principal advantage of online self-supervision is that a robot can adapt to its environment. As opposed to requiring outside supervision and instruction, the robot can learn from its own mistakes. This allows robots equipped with self-supervision to adapt to novel environments or scenarios with little to no human intervention, and is a powerful tool for achieving both robustness and reusability. Not surprisingly, online self-supervised approaches to learning have gained increasing popularity in recent years, especially in the context of the DARPA LAGR program [178].

Approaches for self-supervised online learning can be divided into two distinct classes. The first is near-to-far learning, the goal of which is to learn how to interpret a far-range, lower resolution sensor using a near range, higher-resolution sensor. LAGR robots provide a clear example of the potential utility of this approach, with a monocular camera system that can perceive terrain at an order of magnitude farther range than their stereo camera system. Near-to-far learning is achieved by remembering how specific patches of terrain were perceived by a far range sensor, and then later observing them with a near-range sensor. This provides a correspondence between the output of the two sensing modalities, and provides the necessary data to learn a mapping between the outputs of the far-range and near-range sensors.

Near-to-far approaches serve two key goals: they automate the difficult process of interpreting a far range, low resolution sensor, and they allow this interpretation to adapt online. The previously

a data labeling technique; offline hand labeling could also be used with similar results

[4] This very issue is discussed in [176]; however the proposed approach does not fundamentally address the problem

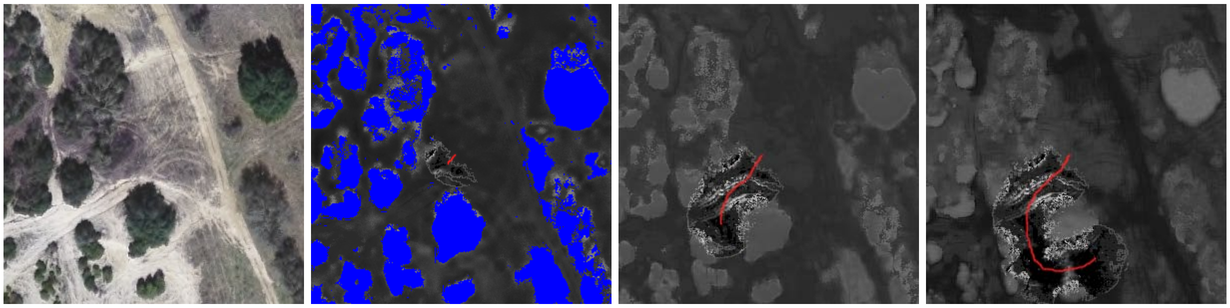


Figure 4.1: An example of near-to-far learning from [104]. In this example, as a robot drives through an environment (left to right) it uses its short range sensors to train the interpretation of a far range sensor (in this case the satellite image at left)

discussed technique used by the Stanford Grand Challenge team [30, 34] is one example, making use of a short range laser to learn the appearance of road in a monocular camera system. Other near-to-far learning implementations include alternate techniques for learning the appearance of roads [179], learning the appearance of different classes (based on a near-range supervised multi-class classification) [148, 180, 181, 182, 183, 184], learning the appearance of traversable terrain (based on a near-range supervised classification of traversability) [174, 185, 186, 187] and even learning terrain costs directly (using costs from a near-range sensor) [103, 104](Figure 4.1). A more complex implementation learned to predict sets of geometric features from far-range sensors (based on the features produced from near-range sensors) [171].

While near-to-far learning can make interpretation of far-range sensors much easier or even automatic, it does not actually solve the core problem of constructing preference models. That is, near-to-far learning still requires a pre-existing, correct interpretation of near-range sensors. In fact, this near-range interpretation is increased in importance, since it is being used as the ground truth signal for learning. For example, when learning to map a near-range to a far-range cost function [103, 104], the assumption is that the near-range cost function has already been properly tuned, an assumption that only holds with an intensive manual tuning effort. Approaches that instead map features or traversability from near-to-far in turn require a cost function defined over the learned far-range interpretation. Therefore, while near-to-far learning can improve robustness and reliability, as well as eliminate some hand tuning problems, it does not address the core problem of completely automating construction of preference models.

The other distinct class of self-supervised learning is learning from proprioception, also called learning from underfoot. As opposed to near-to-far learning, the ground truth signal comes not from a higher resolution exteroceptive sensor, but from proprioceptive sensors. Aside from this distinction, the methodology is quite similar: when the robot drives over a patch of terrain, it recalls how that terrain appeared in its near-range sensors just moments ago. This sets up a correspondence between how terrain appears, and how the robot interacts with it. If the action undertaken by the robot is also taken into account, this allows for correspondences from terrain and action to proprioceptive experience to be learned.

Previous work in learning from proprioception has taken differing approaches. One approach is to attempt to learn the traversability of terrain. If a robot is able to drive through a patch of terrain, then it is labeled as traversable. If a robot becomes stuck over a patch of terrain, is unable to move through an obstacle, or is forced into a dangerous state (e.g. motor currents are too high, roll/pitch approaches stability limits, etc.) then the terrain is labeled as non-traversable. This basic approach has been explored by a number of researchers [180, 181, 188, 189]; however, it does contain serious drawbacks. The first is the same issue as with supervised classification (Section 4.1) into traversable and non-traversable terrain: focusing purely on traversability ignores the possibility of relative preferences amongst traversable terrain, and does not allow other considerations to be taken into account. Additionally, this method of labeling terrain requires explicit robot interaction with non-traversable terrain; the robot must drive into a tree or off a cliff before it can learn that those actions are dangerous. Finally, there is the issue of blame assignment; when the robot encounters a failure, it is not necessarily clear which patches of terrain contributed to the failure, and in what way.

The other approach to learning from proprioception has been to learn to model rather than classify the robot's interactions with terrain. This has taken the form of attempting to learn and predict various terramechanical properties, such as terrain roughness [190], vehicle slip [191, 163, 192, 193], soil cohesion [194] and resistance [170], and vegetation height and stiffness [195, 196, 197]. These predicted terrain properties can then either be used directly to affect robot behavior, or used to improve the accuracy of a physical vehicle simulation [156, 192, 193]. In and of itself this is a very useful result, and can provide much more accurate and robust predictions. However, this approach to learning from proprioception still does not address the fundamental problem of learning relative preferences between various terrains and actions. Instead, as with supervised classification, it simply converts this problem to a new space. Now, instead of determining terrain cost based solely on their appearance, they can be costed based on the robot interaction predicted from appearance. This new proprioceptive feature space may be lower dimensional than the exteroceptive feature space; it may also be more intuitively related to safety or some other metric, or even allow for more informed decision making [192, 193]. Regardless, the fact remains that it will still be necessary to map this space into relative preferences over possible outcomes in order to be useful for planning[5].

There are also some issues that are shared between both approaches to learning from proprioception. Learning from proprioception requires actual robot interaction with a wide variety of terrain. The higher resolution and dimensionality of the proprioceptive properties the robot wishes to predict, the more training data will be necessary. This contradicts the desire for rapid development and deployment of mobile robotic systems. Additionally, these experiences must not just be sampled from safe regions; experience with harsh and dangerous terrain will be necessary in order to learn how to characterize such structures. This allows for the possibility of damage to the robot or its environment, including a catastrophic failure that would permanently cripple it. Especially in mission critical scenarios such as planetary exploration, this possibility cannot be allowed.

Therefore, this work makes the following claim: in order to achieve reliability and rapid development, mobile robotic systems cannot *solely* learn online from scratch; they must be given sufficient capability to ensure safety and reliability in a supervised manner. In a way, this is common sense, and is evident in nature: all animals have some degree of necessary survival instinct at birth, and many are supervised and taught by their parents until they are capable of surviving on their own. In addition, the very notion of a preference model implies some sort of outside expert

[5] This is not to say that proprieceptive metrics are not useful; they are in fact used as important empirical evidence later in this work (Chapters 5 and 6). Rather, the issue lies in combining multiple such metrics into a single measure of preference.

or oracle, and not necessarily a single, objective truth to be discovered.

However this is not to say that online self-supervised learning is not incredibly valuable. Continuing the nature metaphor, higher animals do not learn everything they know from genetics or parents; they continue to learn and adapt from experience. In this way, near-to-far learning can be incredibly useful by only necessitating the development and tuning of near-range perception systems; as long as the robot can ensure safety through just this near-range system, it can experience the world (perhaps in a degraded manner) and learn to interpret its far-range sensors. Additionally, there is also a place for learning from proprioception. The exteroceptive feature space is not necessarily stable with respect to terrain preferences and interactions: between different environments patches of terrain may look similar, but have very different mobility properties. This can happen even within the same environment as it evolves over time (e.g. lighting, weather or seasonal changes). Learning from proprioception can help to map from an unstable, appearance based feature space to a more stable, interaction based appearance space [6]. If terrain and action preferences are expressed in this proprioceptive feature space, they will in turn be more accurate if the mapping from appearance to predicted interaction adapts over time. Therefore, while self-supervised learning does not directly address the problem of choosing preferences over terrain and actions, it can serve as an important complement to an approach that does address this problem.

4.3 Supervision through Explicit Reinforcement

An alternate formalism that combines the notion of expert supervision with learning from experience is *Reinforcement Learning*. In reinforcement learning, the robot is provided with a reward (or penalty) signal based on its behavior (in the case of a mobile robot, where it drives or how it drives). By observing what behaviors maximize reward or minimize penalty, the robot can learn to generalize to desirable behavior that leads to greater rewards both immediately and in the future. Reinforcement learning has proven effective as a way to to learn a control policy ([198] connects reinforcement learning directly to adaptive optimal control). This effectiveness is due to a similar theme as other forms of learning: while it may not be easy to define how to get to a good configuration of a state space, it should be easier to at least define which configurations are good; reinforcement learning can then generalize to a full policy.

Within mobile robotics, applications of reinforcement learning have mostly been successful when applied to the task of learning a control policy under complex dynamics [7, 199] or learning trajectories based on perceptual data [200, 201, 202, 203]. The latter is somewhat similar to the task of constructing a preference model; the catch is that using reinforcement learning in such a context requires a certain amount of structure be built on top of the perceptual data. In addition, a reward function must be defined that expresses which states are better than others. For instance, [200] defines a reward function that favors open areas while balancing turning, while [203] rewards velocities similar to a desired input while penalizing collisions with a defined obstacle class. In other words, the reward function must encode the relative preferences and tradeoffs the designer wishes the robot to exhibit; the reward function is now the preference model. Manual construction of such a reward function will therefore suffer the same pitfalls.

[6] This same issue of stability occurs with near-to-far learning. For example, [171] learns a mapping from near-to-far geometry, and then maps from geometry to cost. Because geometry is more stable, small changes or errors in this near-to-far mapping do not affect cost as significantly as would a direct appearance to cost mapping

An alternative is the idea of sparse rewards [201, 202], where the reward signal is zero in most cases, but positive or negative after specific good or bad events (such as reaching a goal or colliding with an obstacle). However, this still requires an explicit model or classification defining which states deserve reward or penalty (which also is similar to a preference model). In addition, with sparse rewards blame assignment remains a problem. Finally, as with learning from proprioception the robot must actually traverse bad terrain to receive a penalty and learn from it; that is, driving off a cliff to learn it is a cliff is again required.

Rather than specifically defining a model of such sparse rewards, a human expert can instead observe robot behavior and present a reward or penalty when corresponding behavior is observed. This can also provide a measure of safety, as the robot can be stopped (and penalty awarded) before a dangerous event actually occurs. However, this new paradigm requires a human expert to be present while learning is in progress. Since the rewards are sparse, learning will be much slower than with a dense reward function, requiring even more expert time. [202] specifically proposes seeding the learning procedure via the expert controlling the robot's behavior initially, and specifically demonstrating penalty and reward rich areas. However, this is not an efficient use of expert time; if an expert is required to actually operate a robot in order for it to learn, than an approach is needed which learns faster from such demonstration.

4.4 Supervision through Expert Demonstration

A different approach to supervised learning for mobile robots is supervision through demonstration, also known as imitation learning[7]. A key principle of imitation learning is that while it may be very difficult to quantify why a certain behavior is correct, the actual correct behavior is usually known[8] by a human expert. Therefore, rather than asking a human expert to hand tune a system to achieve a desired behavior, the expert can simply demonstrate it and the robot can tune itself to match the demonstration. Learning from demonstration can take many different forms. One form is task-specific imitation learning, where expert demonstration is used to learn how to reliably perform a specific task [204, 205]. However, this specific nature precludes usefulness to mobile robotic systems that must be robust to novel experiences and reusable in different environments; therefore it is not covered in this document (for a complete survey see [206]). Instead, the focus is on generalizable imitation learning, where demonstration in one domain can be generalized into other, novel domains.

Generalizable learning from demonstration has a long history with mobile robotics. The first notable use was the *Autonomous Land Vehicle In a Neural Network*, or ALVINN, system [157]. A part of the original NavLab platform (Section 2.1), ALVINN consisted of a neural network trained to map from an input monocular camera image to a vehicle steering command; in this way ALVINN was trained to drive Navlab within a specified road lane. Other examples of this form of imitation learning are the DAVE system [158], Behavioral Cloning [207, 208, 209], and an application to planetary exploration [210]. These systems are examples of end-to-end learning: the learning task is structured to map directly from raw sensory input to control actions. Other approaches attempt to learn a mapping from input to actions, but explicitly partition this task into multiple learning problems. For example, the MAMMOTH system [211] learned a mapping to

[7] In this document the terms 'learning from demonstration' and 'imitation learning' are used interchangeably

[8] If it is not, then the problem is ill posed

actions not from raw sensor inputs, but from the outputs of supervised classification [160]. Similarly, the SHOSLIF-N system [212] first performs an independent dimensionality reduction and feature extraction before mapping to actions. Other approaches maintain a database of demonstrated actions, and select the most appropriate action from the database based on the current state [213, 214, 215, 216].

These approaches to learning from demonstration all fall into the category of action prediction: given the state of the world, or features of the state of the world, they attempt to predict the action that the human demonstrator would have selected. In this sense the action prediction approach is closely related to supervised classification; if each possible action is treated as a class label, action prediction attempts to determine which action class a given input falls into. They also fall into the realm of model-free approaches; that is, they attempt to map directly to an action without any internal model of how actions are generated (or why an action is appropriate). This is in contrast to model-based approaches, which contain engineered models for producing control actions, and attempt to learn parameters for these models. For example, [217, 218] uses a a reactive model derived from studies of human behavior, and learns parameters for the model based on human demonstration. [176] makes use of the DAMN architecture [53, 54] (Section 2.1), and learns weights for an arbiter that chooses amongst multiple reactive behaviors.

The fundamental problem with pure action prediction is that it is a purely reactive approach to planning. There is no attempt to model or reason about the consequences of a chosen action. Therefore, all relevant information must be encoded in the current state or features of the state. For certain scenarios this is possible; path trackers are a classic example. However, in general the use of purely reactive systems is incredibly difficult for planning long range, goal directed behaviors through complex environments . Therefore, while at times it has been argued that purely reactive approaches are sufficient [219], it has become clear within the mobile robotics community that at least some level of deliberative planning is necessary for reasoning about complex and long range tasks.

Therefore, what is needed is a form of generalizable learning from demonstration that works with deliberative planning. Such a form would have to be model-based, since longer-range deliberative planning results in an incredibly large (and possibly infinite) space of possible plans. *Inverse Optimal Control* (IOC) provides a potential formalism for exactly such learning from demonstration. While optimal control attempts to find a trajectory through a state space that optimizes some metric, inverse optimal control seeks to discover a metric such that a trajectory through a state space is optimal under said metric. Inverse optimal control was first proposed in [220]; over the years it has been extended to linear systems of increasing generality [221].

The problem that inverse optimal control offers a solution to is exactly that which this thesis also seeks to address: learning a metric (that encodes the desired preferences). When manually constructing a cost function over possible terrains and actions, an engineer usually continues to tweak the mapping until the combination of the perception system, the current mapping, and the planning system produce a desired (or at least acceptable) behavior. IOC automates this exact process: it seeks to find a cost function such that demonstrated behavior will be optimal under said cost function. Therefore, the core technical approach of this thesis is to apply inverse optimal control based techniques to the task of learning mappings from features of state-actions pairs to costs. As opposed to a pure state to action mapping, such a feature to cost mapping can generalize as well as the feature representation generalizes.

The advantages of an approach using IOC to learn preference models from demonstration are in stark contrast to the disadvantages of a manual approach listed in Section 3.2

- It requires no intuition or knowledge about how the autonomy sub-system of a robot is constructed. All that is necessary is the proper knowledge of how the robot should behave.
- An automated approach will not suffer as much from diminishing returns; since there is no human interaction involved in the learning phase (once the demonstration phase is complete), learning can continue as long as necessary.
- It is clear when learning has produced at least a locally optimal preference model. In some cases, global optimality can also be guaranteed.
- An automated learning approach can consider a large set of demonstrations at once, and therefore can prevent itself from overfitting to one example at the expense of another.
- Multiple preference models, based on different subsets of possible inputs, can be learned from the same demonstration. Therefore, the amount of necessary human interaction is not affected by the number of different preference models required. Additionally, since these models are learned to reproduce the same demonstrations, they will automatically be compatible (if they need to be fused)
- Whenever a new preference model is required due to a system or component change, no additional human interaction is necessary; previously recorded demonstration can be used to learn the new model.
- Whenever a new preference model is required due to a preference model failure or change in desired behavior, new demonstrations can be quickly added to the training set (including demonstration that specifically addresses the desired changes).

For these reasons, the learning by demonstration paradigm is proposed for automating the construction of preference models, in order to reduce the human interaction previously required for this task. The use of such an approach can not only lead to more rapid development of mobile robotic systems, but also offers the possibility of improving reliability and robustness by constructing potentially better preference models. Finally, the less work that goes into platform specific development and tuning, the more work can go into reusable aspects of mobile robotic systems.

Chapter 5

Learning Terrain Preferences from Expert Demonstration

Since most deliberative planning systems can be represented in the framework of Markov Decision Processes (MDPs), it is in this framework that the most work in applying inverse optimal control to robotics has been performed. The first attempt was the Inverse Reinforcement Learning (IRL) approach of [222]. Since the reinforcement learning framework is well suite to the task of solving optimal control problems for mobile robotic systems, it stands to reason that the inverse might be well suited for solving inverse optimal control problems. Based on this intuition, the IRL approach sought to find a cost function such that a demonstrated behavior was optimal; that is, it was equal or lesser cost than any other trajectory through the state space with the same start and end. However, this original formulation was ill-posed, as there may be a wide variety of cost functions that meet this criteria (for example, the zero everywhere cost function).

The original IRL framework was later modified by [223] into a new approach known as *Apprenticeship Learning*. Apprenticeship learning assumes a linear cost function; for a state-action pair described by a feature vector F, the cost is equal to $w^T F$ for some weight vector w. Under this assumption, the problem can be reformulated in terms of finding a w such that the planned trajectory has equal cumulative feature counts to the demonstrated trajectory (the cumulative feature counts are simply the sum of F at every state along a trajectory). By linearity, if the cumulative feature counts match, then the cumulative costs will match as well. While this new formulation results in fewer trivial solutions, there still may be many policies that match feature counts without actually matching behavior (apprenticeship learning contains no mechanism for explicitly matching demonstrated behavior). Choosing amongst cost functions requires either additional expert input, or a random mixture of cost functions.

The *Maximum Margin Planning* (MMP) framework [224] addressed this problem by reformulating the task of feature count matching as a constrained optimization problem, which includes an upper bound on the error between the demonstrated and produced trajectory. The MMP framework also provides a convex objective function, ensuring that there is a single global solution. While the original MMP formulation also assumes linear cost functions, a similar constrained optimization approach has been extended to non-linear cost functions [225, 226, 10]. The MMP framework has since proven applicable to a variety of real-world robotics applications [225, 227, 228, 229, 230, 231, 232].

Therefore, the MMP framework for performing inverse optimal control is used in this work.

Specifically, the next section builds on the MMP framework to first derive the LEARCH algorithm, allowing the use of nonlinear cost functions in learning from demonstration. Next, the R-LEARCH and DR-LEARCH algorithms are developed[1] for the purpose of learning preferences over terrain types. Chapter 6 then extends these techniques to allow learning preferences over various driving styles and robot actions.

5.1 Constrained Optimization from Expert Demonstration

Maximum Margin Planning frames learning from demonstration as a constrained optimization problem. In decision theory, a utility function is defined as a relative ordering over all possible options. Since most mobile robotic systems encounter an infinite number of possible plans or behaviors, such an explicit ordering is not possible. Instead, possible behaviors are mapped to a scalar cost function, and the cost value defines the ordering. However, this core idea of an ordering over preferences remains useful. If an expert can concretely define a relative ordering over even a small subset of possible plans, this ordering becomes a constraint on the cost function: the costs assigned to each behavior must match the relative ordering. The more information on relative preferences an expert provides, the more the cost function is constrained.

This section first derives the basic MMP algorithm for learning a linear cost function to reproduce expert demonstration. Next the LEARCH algorithm (LEArning to seaRCH) [10] is presented for extending this approach to non-linear cost functions. Extension of the LEARCH algorithm to dynamic and partially known environments is presented, along with adaptations to add robustness to noisy and imperfect expert demonstration. Finally, experimental results are presented in applying these techniques to a complex mobile robotic system operating in rough and unstructured environments.

The full MMP algorithm is defined over general Markov Decision Processes. However, the following derivation is presented with two simplifications. First, as preferences over terrain types in a robotic system correspond to costs defined over locations in the environment, for now cost functions are defined over features of states, as opposed to features of state-action pairs. This simplification does not result in a loss of generality; extension to full state-action pairs is presented in Chapter 6. Second, this derivation is restricted to deterministic MDPs with a set of absorbing states; that is, goal directed path or motion planners. This change is purely for notational simplification, as most mobile robotic systems use planners of this form (including all the planners directly addressed in this work).

5.1.1 Maximum Margin Planning with Linear Cost Functions

Consider a state space $\mathcal{S}$ over which a planner operates (e.g. $\mathcal{S} = \mathbb{R}^2$). A feature space $\mathcal{F}$ is defined over $\mathcal{S}$. That is, for every $x \in \mathcal{S}$, there exists a corresponding feature vector $F_x \in \mathcal{F}$. F_x can be considered as the raw output of a perception system at state x[2]. For the output of a perception system to be used by a planner, it must be mapped to a scalar cost value; Therefore, C is defined as a cost function, $C : \mathcal{F} \to \mathbb{R}^+$. The cost of a state x is $C(F_x)$. For the moment, only linear cost

[1] A version of this chapter has been previously published in [232]

[2] For now it is assumed that the assignment of a feature vector to a state is static; the extension to dynamic assignment is addressed in Section 5.2

functions of the form $C(F) = w^T F$ are considered; the weight vector w completely defines the cost function. Finally, a path P is defined as a sequence of states in $\mathcal{S}$ that lead from a start s to a goal g. The cost of an entire path is simply defined as the sum of the costs of all states along the path, or alternatively the cost of the cumulative feature counts

$$C(P) = \sum_{x \in P} C(F_x) = \sum_{x \in P} w^T F_x = w^T \sum_{x \in P} F_x \tag{5.1.1}$$

Now consider an example path P_e from a start state s_e to a goal state g_e. If this example path is provided via expert demonstration, then its is reasonable to consider applying inverse optimal control; that is, seeking to find a cost function C such that P_e is the optimal path from s_e to g_e. While a single example demonstration does not imply a single cost function, it does constrain the space of cost functions $\mathcal{C}$: only cost functions that consider P_e the optimal path are acceptable [3]. If a regularization term is also added to encourage simple solutions, then the task of finding an acceptable cost function from an example can be phrased as the following constrained optimization problem:

$$\begin{gathered} \text{minimize } \; O(w) = ||w||^2 \\ \text{subject to the constraints} \\ \sum_{x \in \hat{P}} (w^T F_x) \;\geq\; \sum_{x \in P_e} (w^T F_x) \\ \forall \hat{P} \; s.t. \; \hat{P} \neq P_e, \;\; \hat{s} = s_e, \;\; \hat{g} = g_e \end{gathered} \tag{5.1.2}$$

Unfortunately, this optimization has a trivial solution: $w = \vec{0}$. This issue can be addressed by including an additional margin in each constraint; essentially the problem is rephrased as Maximum Margin Structured Classification [228, 233]. A margin not only removes the trivial solution, it improves generalization by ensuring the stability of any solutions.

As opposed to a constant margin, the size of the margin can be scaled [234] based on the similarity between paths; that is the example only needs to be slightly lower cost than a very similar path, but should be much lower cost than a very different path. Similarity between P_e and an arbitrary path P is encoded by a loss function $L(P_e, P)$, or alternatively $L_e(P)$[4]. The definition of the loss function is somewhat application dependant (aside from the requirement that $L(P_1, P_2) = 0 \iff P_1 = P_2$); the simplest form would be to simply consider how many states the two paths share (a Hamming loss). The constrained optimization can now be rewritten as

[3] For now, it is assumed that all P_e are at least near optimal under at least one $C \in \mathcal{C}$; Section 5.3 will relax this assumption

[4] As a path in this context is considered just a sequence of states, the loss function can be defined either over a full path or over a single state.

$$
\begin{gathered}
\text{minimize } \; O(w) = ||w||^2 \\
\text{subject to the constraints} \\
\sum_{x \in \hat{P}} (w^T F_x - L_e(x)) \; \geq \; \sum_{x \in P_e} (w^T F_x) \\
\forall \hat{P} \; s.t. \; \hat{P} \neq P_e, \;\; \hat{s} = s_e, \;\; \hat{g} = g_e
\end{gathered}
\tag{5.1.3}
$$

$$
L_e(x) = \begin{cases} 1 & \text{if } x \in P_e \\ 0 & \text{otherwise} \end{cases}
$$

Depending on the state space, and the distance from s_e to g_e, there are likely to be an infeasible (and possibly infinite) number of constraints; one for each alternate path to the demonstrated example. However, it is not necessary to enforce every constraint. For any candidate cost function, there is a minimum cost path between any two waypoints, P_*. It is only necessary to enforce the constraint for P_*, as once it is satisfied by definition all other constraints will be satisfied. With this single constraint, (5.1.3) becomes

$$
\begin{gathered}
\text{minimize } \; O(w) = ||w||^2 \\
\text{subject to the constraint} \\
\sum_{x \in P_*} (w^T F_x - L_e(x)) \; \geq \; \sum_{x \in P_e} (w^T F_x) \\
P_* = \arg\min_{P} \sum_{x \in P} (w^T F_x - L_e(x))
\end{gathered}
\tag{5.1.4}
$$

It may not always be possible to exactly meet this constraint (the margin may make it impossible). Therefore, a slack term ζ is added to allow for this possibility

$$
\begin{gathered}
\text{minimize } \; O(w) = \lambda ||w||^2 + \zeta \\
\text{subject to the constraint} \\
\sum_{x \in P_*} (w^T F_x - L_e(x)) \; - \; \sum_{x \in P_e} (w^T F_x) + \zeta \geq 0
\end{gathered}
\tag{5.1.5}
$$

The slack term ζ accounts for the error in meeting the constraint, while λ balances the tradeoff in the objective between regularization and meeting the constraint. Due to the scaling of the margin by the loss function, a concrete relationship can be shown between minimizing this objective, and minimizing the loss

Theorem 5.1.1. *The aggregate loss of P_*, $L_e(P_*) = \sum_{x \in P_*} L_e(x)$, is bounded by ζ*

Proof. Consider the following cases

Case 1 $P_* = P_e$

Then trivially $\zeta = L_e(P_*) = 0$

Case 2 $P_* \neq P_e$

The constraint in (5.1.5) can be rewritten as

$$\sum_{x \in P_*}(w^T F_x) \; - \; \sum_{x \in P_e}(w^T F_x) \geq \sum_{x \in P_*}(L_e(x)) - \zeta$$

By definition $\sum\limits_{x \in P_*}(w^T F_x) \leq \sum\limits_{x \in P_e}(w^T F_x)$ which implies

$$\begin{aligned} &\implies \sum_{x \in P_*}(w^T F_x) \; - \; \sum_{x \in P_e}(w^T F_x) \leq 0 \\ &\implies \sum_{x \in P_*}(L_e(x)) - \zeta \leq 0 \\ &\implies \sum_{x \in P_*}(L_e(x)) \leq \zeta \end{aligned}$$

□

Since ζ is in the objective to be minimized, performing the MMP optimization will also reduce the aggregate loss.

The constraint in (5.1.5) can alternatively be rearranged as

$$\zeta \geq \sum_{x \in P_e}(w^T F_x) \; - \; \sum_{x \in P_*}(w^T F_x - L_e(x)) \tag{5.1.6}$$

Since ζ is in the minimizer, it will always be tight against this constraint that is ζ will always be exactly equal to the difference in (loss augmented) path costs. Therefore, ζ can be replaced in the objective by the constraint in (5.1.6), resulting in the following (unconstrained) optimization problem

$$\begin{gathered} \text{minimize } \; O(w) = \lambda ||w||^2 + \\ \sum_{x \in P_e}(w^T F_x) \; - \; \sum_{x \in P_*}(w^T F_x - L_e(x)) \end{gathered} \tag{5.1.7}$$

or alternatively

$$\begin{gathered} \text{minimize } \; O(w) = \lambda ||w||^2 + \\ \sum_{x \in P_e}(w^T F_x) \; - \; \min_{\hat{P}} \left[\sum_{x \in \hat{P}}(w^T F_x - L_e(x)) \right] \end{gathered} \tag{5.1.8}$$

The final optimization seeks to minimize the difference in cost between the example path P_e and the (loss augmented) optimal path P_*, subject to regularization. $O(w)$ is convex, but non-differentiable; therefore, instead of gradient descent, it can be minimized using the sub-gradient

Algorithm 1: The linear MMP algorithm

```
Inputs : Example Paths $P_e^1, P_e^2, ..., P_e^n$, Feature Map $\mathcal{F}$
$w_0 = \vec{0}$;
for $j = 1...K$ do
    $\mathcal{M} =$ buildCostmap($w_{j-1}, \mathcal{F}$);
    $F_e = F_* = \vec{0}$;
    foreach $P_e^i$ do
        $P_*^i =$ planLossAugmentedPath(start($P_e^i$),goal($P_e^i$),$\mathcal{M}$);
        foreach $x \in P_e^i$ do
            $F_e + = F_e + F_x$;
        foreach $x \in P_*^i$ do
            $F_* = F_* + F_x$;
    $w_j = w_{n-1} + \eta_i[F_* - F_e - \lambda w_{j-1}]$;
    enforcePositivityConstraint($w_j$,$\mathcal{F}$);
return $w_K$
```

method with learning rate η. The sub-gradient of O with respect to w is

$$\nabla O = 2\lambda w + \sum_{x \in P_e} F_x \ - \sum_{x \in P_*} F_x \tag{5.1.9}$$

Intuitively, (5.1.9) says that the direction that will most minimize the objective function is found by comparing feature counts. If more of a certain feature is encountered on the example path than the current minimum cost path P_*, the weight on that feature (and therefore the cost) should be decreased. Likewise, if less of a feature is encountered on the example path than on P_*, the weight should be increased. Although the margin does not appear in the final sub-gradient, it does affect the computation of P_* (a key difference between MMP and apprenticeship learning [223]). The final linear MMP algorithm consists of iteratively computing feature counts and then updating the cost function until convergence. One additional caveat is to ensure that only cost functions that map to $\mathbb{R}^+$ are considered (a requirement of most motion and path planners).This is achieved by identifying F such that $w^T F \leq 0$, and projecting w back into the space of allowable cost functions.

The MMP framework easily supports multiple example paths. Each example implies its own constraints as in (5.1.4), its own objective as in (5.1.8), and its own sub-gradient as in (5.1.9). Updating the cost weights can take place either on a per example basis, or the feature counts can be computed in a batch with a single update. The latter is computationally preferable, as it may result in fewer cost function evaluations, and projections back into the space of allowable cost functions. In the (likely) case that no cost function can perfectly satisfy all the constraints from multiple examples, the optimization will converge to a consensus cost function that balances the preferences implied by each path. The final linear MMP algorithm is presented in Algorithm 1.

5.1.2 MMP with Non-Linear Cost Functions

The derivation to this point has assumed that the space of possible cost functions $\mathcal{C}$ consists of all functions of the form $C(F) = w^T F$. Extension to other, more descriptive spaces of cost functions is possible by considering (5.1.8) for any cost function C

$$\text{minimize } \mathcal{O}[C] = \lambda\textsc{Reg}(C) + \sum_{x \in P_e} C(F_x) - \min_{\hat{P}} \left[\sum_{x \in \hat{P}} (C(F_x) - L_e(x)) \right] \tag{5.1.10}$$

$\mathcal{O}[C]$ is now an objective functional over a cost function, and Reg represents a regularization functional. We can now consider the sub-gradient in the space of cost functions

$$\nabla\mathcal{O}_F[C] = \lambda\nabla\textsc{Reg}_F[C] + \sum_{x \in P_e} \delta_F(F_x) - \sum_{x \in P_*} \delta_F(F_x) \tag{5.1.11}$$

$$P_* = \arg\min_P \sum_{x \in P} (C(F_x) - L_e(x))$$

where δ is the Dirac delta at the point of evaluation. Simply speaking, the functional gradient is positive at values of F corresponding to states in the example path, and negative at values of F corresponding to states in the current planned path. If the paths both contain a state corresponding to F, their contributions cancel.

Applying gradient descent directly in this space would result in an extreme form of overfitting; essentially, it would involve raising or lowering the cost associated with specific values of F encountered on either path, and would therefore involve no generalization whatsoever. Instead, a different space of cost functions is considered

$$\begin{aligned} \mathcal{C} =& \{C \mid C = \sum_i \eta_i R_i(F), \quad R_i \in \mathcal{R}, \quad \eta_i \in \mathbb{R}\} \\ \mathcal{R} =& \{R \mid R : \mathcal{F} \to \mathbb{R} \ \wedge \ \textsc{Reg}(R) < \nu\} \end{aligned} \tag{5.1.12}$$

$\mathcal{C}$ is now defined as the space of weighted sums of functions $R_i \in \mathcal{R}$, where $\mathcal{R}$ is a space of functions of limited complexity that map from the feature space to a scalar. Choices of $\mathcal{R}$ include linear functions, parametric functions, neural networks, decision trees, etc. As in gradient boosting [235], this space represents a limited set of 'directions' for which a small step can be taken; the choice of the direction set in turn controls the complexity of $\mathcal{C}$.

With this new definition, a gradient descent update takes the form of projecting the functional gradient[5] onto the direction set by finding the element $R_* \in \mathcal{R}$ that maximizes the inner product $\langle -\nabla\mathcal{O}_F[C], R_* \rangle$. The maximization of the inner product between the functional gradient and the

[5] For the moment, the regularization term is ignored.

hypothesis space can be understood as a learning problem:

$$\begin{aligned} R_* &= \arg\max_R \langle -\nabla \mathcal{O}_F[C], R \rangle \\ &= \arg\max_R \sum_{x \in P_e \cap P_*} -\nabla \mathcal{O}_F[C] R(F_x) \\ &= \arg\max_R \sum_{x \in P_e \cap P_*} \alpha_x y_x R(F_x) \\ \alpha_x &= | \bigtriangledown \mathcal{O}_{F_x}[C] | \quad y_x = -\text{sgn}(\bigtriangledown \mathcal{O}_{F_x}[C]) \end{aligned} \tag{5.1.13}$$

In this form, it can be seen that finding the projection of the functional gradient involves solving a weighted classification problem; the element of $\mathcal{R}$ that best discriminates between features vectors for which the cost should be raised or lowered maximizes the inner product. Alternatively, defining $\mathcal{R}$ as a class of regressors adds an additional regularization to each individual R_* [10]. Intuitively, the regression targets y_x are positive in regions of the feature space that the planned path visits more than the example path (indicating a desire to raise the cost), and negative in regions that the example path visits more than the planned path. Each regression target is weighted by a factor α_x based on the magnitude of the functional gradient.

In comparison to the linear MMP formulation, this approach can be understood as trying to minimize the error in *visitation counts* instead of feature counts. For a given feature vector F and path P, the visitation count U is the cumulative count of the number of states $x \in P$ such that $F_x = F$. The visitation counts can be split into positive and negative components, corresponding to the current planned and example paths. Formally

$$\begin{aligned} U_+(F) &= \sum_{x \in P_*} \delta_F(F_x) \\ U_-(F) &= \sum_{x \in P_e} \delta_F(F_x) \\ U(F) &= U_+ - U_- = \sum_{x \in P_*} \delta_F(F_x) \quad - \sum_{x \in P_e} \delta_F(F_x) \end{aligned} \tag{5.1.14}$$

Comparing this formulation to (5.1.11) demonstrates that the planned visitation counts minus the example visitation counts equals the negative functional gradient (ignoring regularization). This allows for the computation of regression targets and weights purely as a function of the visitation counts, providing a straight forward implementation (Algorithm 2).

A final addition to this algorithm involves a slightly different approach to optimization. Gradient descent can be understood as encouraging functions that are ’small’ in the l_2 norm; by controlling the learning rate η and the number of epochs, it is possible to constrain the complexity of the learned cost function. However, instead we can consider *exponentiated functional gradient descent*, which is a generalization of exponentiated gradient to functional gradient descent [10]. Exponentiated functional gradient descent encourages functions that are ‘sparse’ in the sense of having many small values and a few potentially large values. This change results in $\mathcal{C}$ being redefined as

$$\mathcal{C} = \{ C \mid C = e^{\sum_i \eta_i R_i(F)}, \quad R_i \in \mathcal{R}, \quad \eta_i \in \mathbb{R} \} \tag{5.1.15}$$

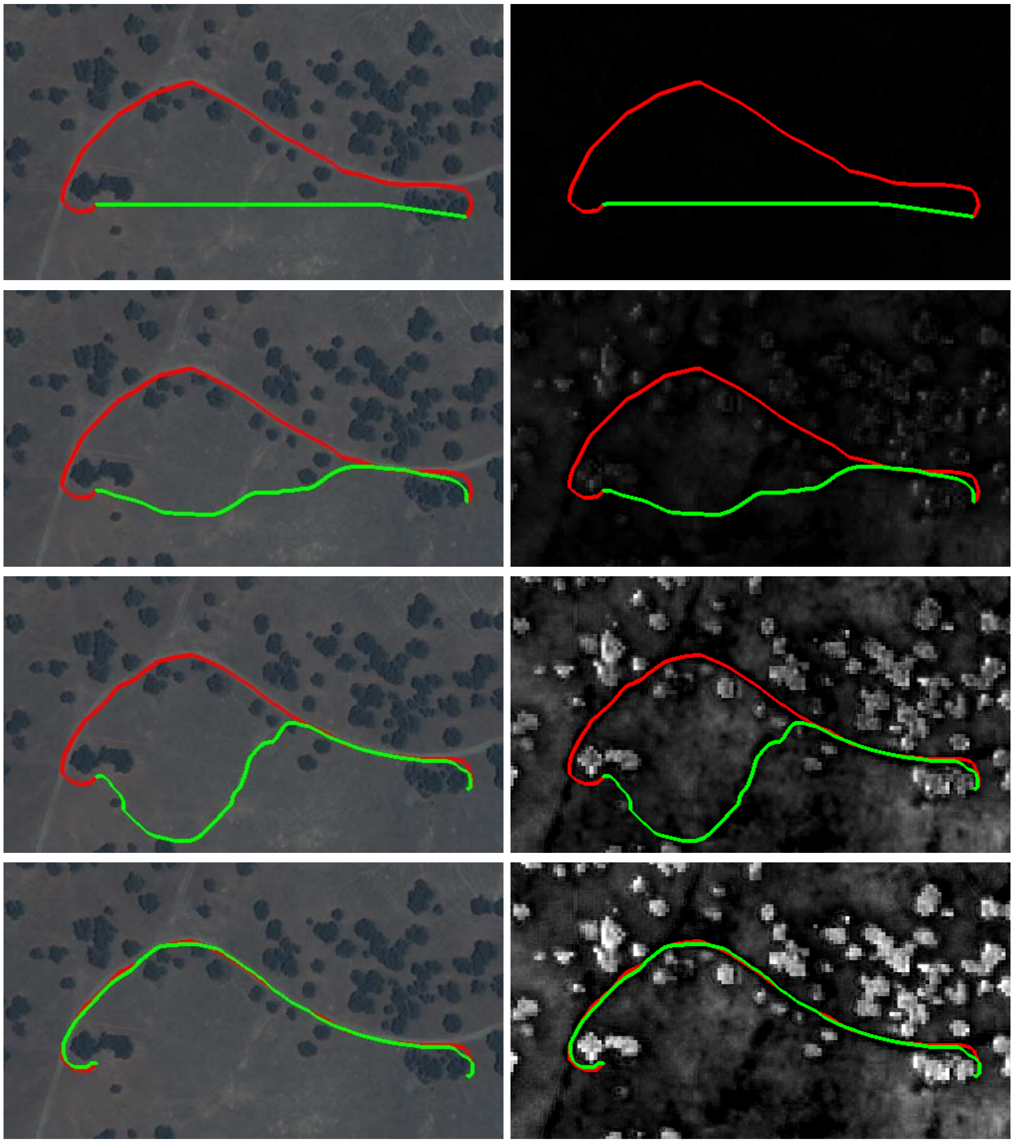


Figure 5.1: An example of the LEARCH algorithm learning to interpret satellite imagery (Left) as costs (Right). Brighter pixels indicate higher cost. As the cost function evolves (top to bottom), the current plan (green) recreates more and more of the example plan (red). Quickbird imagery courtesy of Digital Globe, Inc. Images cover approximately 300 m X 250 m.

Another beneficial effect of this redefined space is that $\mathcal{C}$ naturally maps to $\mathbb{R}^+$ without any need for projecting the result of each gradient descent update into the space of valid cost functions. This final algorithm for non-linear inverse optimal control is called LEARCH [226, 10, 229, 232] and is presented in Algorithm 2. An example of the algorithm in action is presented in Figure 5.1

It should be noted that while seemingly similar, LEARCH is fundamentally different from supervised classification approaches presented in [30, 176]. While examples of 'good' terrain are collected in a similar manner, these approaches simply assume that terrain near where the vehicle was demonstrated driving should be labeled as 'bad'; the assumption is that a classifier will be

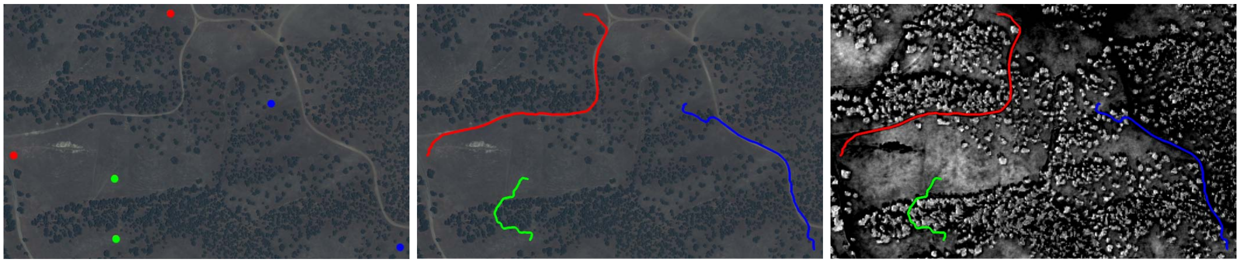


Figure 5.2: Generalization of the LEARCH algorithm. The cost function learned from the single example in Figure 5.1 generalizes over terrain never seen during training (shown at approximately 1/2 scale) resulting in similar planner behavior. 3 sets of waypoints (Left) are shown along with the corresponding paths (Center) planned under the learned cost function (Right).

Algorithm 2: The LEARCH algorithm

```
Inputs : Example Paths $P_e^1, P_e^2, ..., P_e^n$, Feature Map $\mathcal{F}$
$C_0 = 1$;
for $j = 1...K$ do
    $\mathcal{M} = $ buildCostmap($C_{j-1}, \mathcal{F}$);
    $U_+ = U_- = \vec{0}$;
    foreach $P_e^i$ do
        $P_*^i = $ planLossAugmentedPath(start($P_e$),goal($P_e$),$\mathcal{M}$);
        foreach $x \in P_e^i$ do
            $U_-(\mathcal{F}_x) = U_-(\mathcal{F}_x) + 1$;
        foreach $x \in P_*^i$ do
            $U_+(\mathcal{F}_x) = U_+(\mathcal{F}_x) + 1$;
    $T_f = T_o = T_w = \emptyset$;
    $U = U_+ - U_-$;
    foreach $\mathcal{F}_x$ such that $U(\mathcal{F}_x) \neq 0$ do
        $T_f = T_f \bigcup \mathcal{F}_x$;
        $T_o = T_o \bigcup$ sgn($U(\mathcal{F}_x)$);
        $T_w = T_w \bigcup |U(\mathcal{F}_x)|$;
    $R_j = $ trainWeightedRegressor($T_f, T_o, T_w$);
    $C_j = C_{j-1} * e^{\eta_j R_j}$;
return $C_K$
```

able to deal with the noisy labeling. In contrast, LEARCH determines a set of states for which the total cost *must* be increased; otherwise the demonstration would have traversed through those states. Essentially, negative examples of where to drive are implied by where the demonstrator explicitly chose not to drive, rather than simply nearby regions that the demonstrator could have driven through. Additionally, the terrain that the demonstration traversed is not explicitly considered as 'good' terrain; rather its costs are only lowered until the path is preferred (for specific waypoints);there could still be high cost regions along it. This distinction allows LEARCH to generalize well over areas for which it was not explicitly trained (Figure 5.2).

5.2 Extension to Dynamic and Unknown Environments

The previous derivation of MMP and LEARCH only considered the scenario where the mapping from states to features is static and fully known a priori. In this section, the ideas of [225] are applied to extend the LEARCH algorithm to the scenario where neither of these assumptions holds, such as when features are generated from a mobile robot's perception system. The limited range inherent in onboard sensing implies a great deal of the environment may be unknown; for truly complex navigation tasks, the distance between waypoints is generally at least one or two orders of magnitude larger than the sensor range. Further, changing range and point of view from environmental structures means that even once an object is within range, its perceptual features are continually changing. Finally, there are the actual dynamics of the environment: objects may move, lighting and weather conditions can change, etc.

Since onboard perceptual inputs are not static, a robot's current plan must also be continually recomputed. The original MMP constraint must be altered in the same way: rather than enforcing the optimality of the entire example behavior once, the optimality of all example behavior must be continually enforced as the current plan is recomputed. Formally, we add a time index t to account for dynamics. F_x^t represents the perceptual features of state x at time t. P_e^t represents the example behavior starting from the current state at time t to the goal, with associated loss function L_e^t. The objective becomes

$$\text{minimize} \;\; \mathcal{O}[C] = \; \lambda\textsc{Reg}(C) \; + \sum_t \left(\sum_{x \in P_e^t} C(F_x^t) \; - \; \min_{\hat{P}^t} \left[\sum_{x \in \hat{P}^t} (C(F_x^t) - L_e^t(x)) \right] \right) \tag{5.2.1}$$

the new functional gradient is

$$\nabla \mathcal{O}_F[C] = \sum_t \left(\sum_{x \in P_e^t} \delta_F(F_x^t) \;\; - \;\; \sum_{x \in P_*^t} \delta_F(F_x^t) \right) \tag{5.2.2}$$

$$P_*^t = \arg\min_{P^t} \left[\sum_{x \in P^t} (C(F_x^t) - L_e^t(x)) \right]$$

The cost function C does not have a time index: the optimization is searching for the single cost function that best reproduces example behavior over an entire time sequence.

It is important to clarify exactly what P_e^t represents. Until now, the terms *plan* and *behavior* have been interchangeable. This is true in the static case since the environment never evolves; as long as a plan is sufficiently followed, it does not need to be recomputed. However, in the dynamic case, an expert's plan and behavior are different notions: the plan is the currently intended future behavior, and the behavior is the result of previous plans. Therefore, P_e^t would ideally be the expert's *plan* at time t, not example behavior from time t onwards.

However, this information is generally not available: it would require the recording of an expert's instantaneous plan at each point in time. Even if a framework for such a data collection were to be implemented, it would turn the collection of training data into an extremely tedious and expensive process. Therefore, in practice we approximate the current plan of an expert P_e^t with the expert's behavior from t onwards. Unfortunately, this approximation can potentially create situations where the example at certain timesteps is suboptimal or inconsistent. The consequences of this inconsistency and possible solutions are discussed in Section 5.3 (see Figure 5.5).

Once dynamics have been accounted for, the limited range of onboard sensing can be addressed. At time t, there may be no perceptual features available corresponding to the (potentially large) section of the example path that is outside of current perceptual range. In order to perform long range navigation, a mobile robotic system must already have some approach to planning through terrain it has not directly sensed. Solutions include the use of prior knowledge [105, 229], extrapolation from recent experience [236, 31], or simply to assume uniform properties of unknown terrain.

Therefore, we define the set of visible states at time t as $\mathcal{V}^t$. The exact definition of visible depends on the specifics of the underlying robotic system's data fusion: $\mathcal{V}^t$ should include all states for which the cost of state x at time t is computed with the cost function currently being learned, C[6]. For all other states $\bar{\mathcal{V}}^t$, we can assume the existence of some alternate function for computing cost, $C_{\bar{\mathcal{V}}}(x)$; again this could be as simple as a constant.

Since we have explicitly defined $\mathcal{V}^t$ as the set of states at time t for which C is the correct mechanism for computing cost, the cost of a general path P^t is now computed as

$$\sum_{x \in P^t \cap \mathcal{V}^t} C(F_x^t) \;+\; \sum_{x \in P^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x)$$

It is important to note that $C_{\bar{\mathcal{V}}}(x)$ is not dependent on F_x^t. With this new formulation for the cost of a path, the objective functional becomes

$$\begin{aligned}\text{minimize } \mathcal{O}[C] = \;& \lambda \textsc{Reg}(C) \\ +\; & \sum_t \left(\sum_{x \in P_e^t \cap \mathcal{V}^t} C(F_x^t) \;+\; \sum_{x \in P_e^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right) \\ -\; & \sum_t \min_{\hat{P}^t} \left(\sum_{x \in \hat{P}^t \cap \mathcal{V}^t} (C(F_x^t) - L_e^t(x)) \;+\; \sum_{x \in \hat{P}^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right)\end{aligned} \tag{5.2.3}$$

with functional gradient

$$\begin{aligned}\nabla \mathcal{O}_F[C] &= \sum_t \left(\sum_{x \in P_e^t \cap \mathcal{V}^t} \delta_F(F_x^t) \;-\; \sum_{x \in P_*^t \cap \mathcal{V}^t} \delta_F(F_x^t) \right) \\ P_*^t &= \arg\min_{\hat{P}^t} \left[\sum_{x \in \hat{P}^t \cap \mathcal{V}^t} (C(F_x^t) - L_e^t(x)) \;+\; \sum_{x \in \hat{P}^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right]\end{aligned} \tag{5.2.4}$$

since the gradient is computed with respect to C, it is only nonzero for visible states; the $C_{\bar{\mathcal{V}}}(x)$ terms disappear. However, $C_{\bar{\mathcal{V}}}$ still factors into the planned behavior, and therefore does affect the learned component of the cost function. Just as LEARCH learns C to recreate desired behavior when using a specific planner, it learns C to recreate behavior when using a specific $C_{\bar{\mathcal{V}}}$. However,

[6] For instance, all locations that are within current sensor range and have been observed by said sensors.

if the example behavior is inconsistent with $C_{\bar{\mathcal{V}}}$, it will be more difficult for the planned behavior to match the example. Such an inconsistency could occur if the expert has different prior knowledge than the robot, or interprets the same knowledge differently. Inconsistency can also occur due to the previously discussed mismatch between expert plans and expert behavior. A solution to this problem is discussed in Section 5.3.

The projection of the functional gradient onto the hypothesis space becomes

$$R_* = \arg\max_R \sum_t \left(\sum_{x \in (P_e \cup P_*) \cap \mathcal{V}^t} \alpha_x^t y_x^t R(F_x^t) \right) \tag{5.2.5}$$

Contrasting the final form for R_* with that of (5.1.13) helps to summarize the changes that result in the LEARCH algorithm for dynamic environments. Specifically, a single expert demonstration from start to goal is discretized by time, with each timestep serving as an example of what behavior to plan given all data to that point in time. For each of these discretized examples, only visitation counts in visible states are used. The resulting Dynamic LEARCH (D-LEARCH) algorithm is presented in Algorithm 3.

A final detail for a D-LEARCH implementation is the source of the input perceptual features. Rather than computing and logging these features online, it is useful to record all raw sensor data, and then to compute the features by simulating perception offline. This allows existing expert demonstration to be reused if new feature extractors are added, or existing ones modified; perception is simply re-simulated to produce the new inputs. In this way, learning a cost function when the perception system is modified requires no additional expert interaction.

5.3 Imperfect and Inconsistent Demonstration

The MMP framework implicitly assumes that one or more cost functions exist under which demonstrated behavior is near optimal. Generally this is not the case, as there will always be noise in human demonstration Further, multiple examples possibly collected from different environments and different experts may be inconsistent with each other (due to inconsistency in human behavior, a different concept of what is desirable, or an incomplete perceptual description of the environment by the robot). Finally, sometimes experts are flat out wrong, and demonstrate behavior that is not even close to optimal (under nearly any definition).

While the MMP framework is robust to poor training data, it does suffer degraded overall performance and generalization (in the same way that supervised classification performance is degraded by noisy or mislabeled training data). Attempting to have an expert sanitize the training input after initial demonstration is disadvantageous for two reasons. First it creates an additional need for human involvement, eliminating much of the time savings promised by this approach. Second, it assumes that an expert can detect all errors; while this may be true for extreme cases, a human expert is no more capable of identifying small amounts of noise than he is of preventing that noise in the first place. Even if detecting and filtering out noisy demonstration is automated (as in Section 5.3.3), removing any and all noisy or imperfect demonstration would remove a large percentage of available training data. This would greatly increase the amount of effort that must be expended to produce a viable training set; it may also remove example demonstrations from which

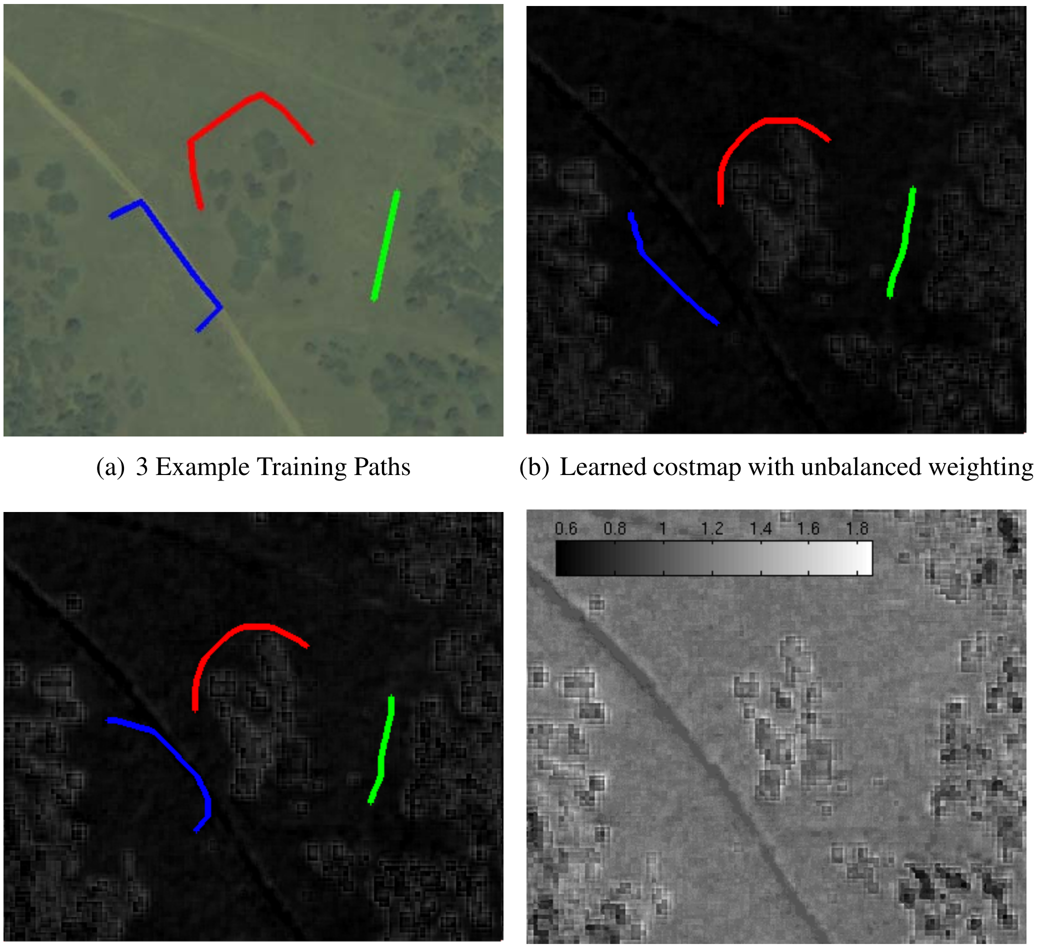
Algorithm 3: The Dynamic LEARCH algorithm

```
Inputs : Example Behaviors $P_e^1, P_e^2, ..., P_e^n$, Sensor Histories $\mathcal{H}^1, \mathcal{H}^2, ..., \mathcal{H}^n$, Cost Map $C_{\bar{\mathcal{V}}}$
$C_0 = 1$;
foreach $P_e^i$ do
    for $\tau$ =firstTime($P_e^i$): $\Delta\tau$ :lastTime($P_e^i$) do
        $P_e^{\tau,i}$ = extractPathSegment($P_e^i$,$\tau$,lastTime($P_e^i$));
        [$\mathcal{F}^{\tau,i}, \mathcal{V}^{\tau,i}$] = simulatePerception($\mathcal{H}^i$,firstTime($P_e^i$),$\tau$);
for $j = 1...K$ do
    $U +_= U_- = \vec{0}$;
    foreach $P_e^{t,i}$ do
        $\mathcal{M}^{t,i}$ = buildCostmap($C_{i-1}$,$\mathcal{F}^{t,i}$,$\mathcal{V}^{t,i}$,$C_{\bar{\mathcal{V}}}$);
        $P_*^{t,i}$ = planLossAugmentedPath(start($P_e^{t,i}$),goal($P_e^{t,i}$),$\mathcal{M}^{t,i}$);
        foreach $x \in P_e^{t,i} \bigcap \mathcal{V}^{t,i}$ do
            $U_-(\mathcal{F}_x^{t,i}) = U_-(\mathcal{F}_x^{t,i}) + 1$;
        foreach $x \in P_*^{t,i} \bigcap \mathcal{V}^{t,i}$ do
            $U_+(\mathcal{F}_x^{t,i}) = U_+(\mathcal{F}_x^{t,i}) + 1$;
    $T_f = T_o = T_w = \emptyset$;
    $U = U_+ - U_-$;
    foreach $\mathcal{F}_x^{t,i}$ such that $U(\mathcal{F}_x^{t,i}) \neq 0$ do
        $T_f = T_f \bigcup \mathcal{F}_x^{t,i}$;
        $T_o = T_o \bigcup$ sgn($U(\mathcal{F}_x^{t,i})$);
        $T_w = T_w \bigcup |U(\mathcal{F}_x^{t,i})|$;
    $R_j$ = trainWeightedRegressor($T_f, T_o, T_w$);
    $C_j = C_{j-1} * e^{\eta_j R_j}$;
return $C_K$
```

something could still have been learned. Therefore, a practical and robust learning approach must be able to handle a reasonable amount of error in provided demonstration. This section describes modifications to the LEARCH algorithm that can increase robustness and improve generalization in the face of noisy, imperfect or inconsistent expert demonstration.

5.3.1 Unachievable Example Behaviors

Experts do not necessarily plan their example behavior in a manner consistent with a robot's planning system: this assumption is not part of the MMP framework. However, what is assumed is that there exists at least one allowable cost function that will cause the robot's planner to reproduce demonstrated behavior (by scoring said behavior as the minimum cost plan). Unfortunately, this is not always the case: it is possible for an example to be *unachievable*. An unachievable example is defined as one such that no consistent cost function, when applied to the available perceptual feature representation, will result in the specified planning system reproducing the example demonstration. For example, an expert may give an inconsistently wide berth to obstacles, or make wider turns than are necessary. Perhaps the most intuitive single case is if an expert turns



(a) 3 Example Training Paths

(b) Learned costmap with unbalanced weighting

(c) Learned costmap with balanced weighting

(d) Ratio of the balanced to unbalanced costmaps

Figure 5.3: The red path is an unachievable example path, as it will be less expensive under any cost function to cut more directly across the grass. With standard unbalanced weighting (b), the unachievable example forces down the cost of grass, and prevents the blue example from being achieved. Balanced weighting (c) prevents this bias, and the blue example is achieved. Overall, grass is approximately 12% higher cost with balanced than unbalanced weighting (d)

left around a large obstacle, when turning right would have been slightly shorter. The consequence is that example paths often take slightly longer routes through similar terrain than are optimal (for any consistent costing of similar terrain) [229, 231, 232]; depending on planner details (such as c-space expansion and dynamic constraints) such examples are often unachievable.

It is instructive to observe what happens to the functional gradient with an unachievable example. Imagine a section of an environment where all states are described by the identical feature vector F'. Under this scenario, (5.1.11) reduces to

$$\nabla \mathcal{O}_{F'}[C] = \begin{cases} \sum_{x \in P_e} 1 \quad - \quad \sum_{x \in P_*} 1 & \text{if } F = F' \\ 0 & \text{if } F \neq F' \end{cases}$$

The functional gradient depends only on the lengths of the example and current plan, independent of the cost function. If the paths are not of equal length, then the optimization will never be satisfied. Specifically, if the example path is too long, there will always be an extra component of the gradient that attempts to lower costs at F'. Intuitively, an unachievable example implies that the cost of certain terrain should be 0, as this would result in any path through that region being optimal. However, since costs are constrained to $\mathbb{R}^+$, this will never be achieved. Instead an unachievable example will have the effect of unnecessarily lowering costs over a large section of the feature space, and artificially reducing dynamic range. Depending on the expressiveness of $\mathcal{R}$, an unachievable example counteracts the constraints of other (achievable) paths, resulting in poorer performance and generalization (see Figure 5.3).

This negative effect can be avoided by performing a slightly different regression or classification when projecting the gradient. Instead of minimizing the weighted error, the *balanced* weighted error is minimized; that is, both positive and negative targets make an equal sum contribution. Formally, in (5.2.5) R_* is replaced with R_*^B defined as

$$\begin{aligned} R_*^B &= \arg\max_R \sum_t \left(\sum_{y_x^t > 0} \frac{\alpha_x^t R(F_x^t)}{N_+} - \sum_{y_x^t < 0} \frac{\alpha_x^t R(F_x^t)}{N_-} \right) \\ N_+ &= \sum_t \sum_{y_x^t > 0} \alpha_x^t = |U_+|_1 \qquad N_- = \sum_t \sum_{y_x^t < 0} \alpha_x^t = |U_-|_1 \end{aligned} \tag{5.3.1}$$

In the extreme unachievable case described above, R_*^B will be zero everywhere; the optimization will be satisfied with the cost function as is. The effect of balancing in the general case can be observed by rewriting the regression operation in terms of the planned and example visitation counts, and observing the correlation of their inputs.

$$R_* = \arg\max \langle R, U_+ - U_- \rangle \quad R_*^B = \arg\max \langle R, \frac{U_+}{N_+} - \frac{U_-}{N_-} \rangle$$

Theorem 5.3.1. *The regression targets of R_* and R_*^B are always correlated, except when the visitation counts between the example and planned path are perfectly correlated.*

Proof.

$$\begin{aligned} \langle U_+ - U_-, \frac{U_+}{N_+} - \frac{U_-}{N_-} \rangle &= \frac{\langle U_+, U_+ - U_- \rangle}{N_+} + \frac{\langle U_-, U_- - U_+ \rangle}{N_-} \\ &= \frac{\langle U_+, U_+ \rangle}{N_+} - \frac{\langle U_+, U_- \rangle}{N_+} + \frac{\langle U_-, U_- \rangle}{N_-} - \frac{\langle U_+, U_- \rangle}{N_-} \\ &= \frac{|U_+|^2}{N_+} + \frac{|U_-|^2}{N_-} - \frac{\langle U_+, U_- \rangle}{N_+} - \frac{\langle U_+, U_- \rangle}{N_-} \\ &= \frac{|U_+|^2}{N_+} + \frac{|U_-|^2}{N_-} - (\frac{1}{N_+} + \frac{1}{N_-}) \langle U_+, U_- \rangle \end{aligned} \tag{5.3.2}$$

By the Cauchy-Schwarz inequality, $\langle U_+, U_- \rangle$ is bounded by $|U_+||U_-|$, and is only tight against this bound when the visitation counts are perfectly correlated, which implies

$$\langle U_+, U_- \rangle = |U_+||U_-| \iff U_- = \kappa U_+ \implies |U_-| = \kappa |U_+| \ , \ N_- = \kappa N_+$$

for some scalar κ. By substitution

$$\begin{aligned}
\frac{|U_+|^2}{N_+} + \frac{|U_-|^2}{N_-} - (\frac{1}{N_+} + \frac{1}{N_-})\langle U_+, U_- \rangle &\geq \frac{|U_+|^2}{N_+} + \frac{|U_-|^2}{N_-} - (\frac{1}{N_+} + \frac{1}{N_-})|U_+||U_-| \\
&= \frac{|U_+|^2}{N_+} + \frac{\kappa^2|U_+|^2}{\kappa N_+} - (\frac{1}{N_+} + \frac{1}{\kappa N_+})\kappa|U_+||U_+| \\
&= \frac{|U_+|^2}{N_+} + \frac{\kappa|U_+|^2}{N_+} - \frac{|U_+|^2}{N_+} - \frac{\kappa|U_+|^2}{N_+} \\
&= 0
\end{aligned}$$

When $\langle U_+, U_- \rangle$ is not tight against the upper bound

$$\langle U_+ - U_-, \frac{U_+}{N_+} - \frac{U_-}{N_-} \rangle \geq 0$$

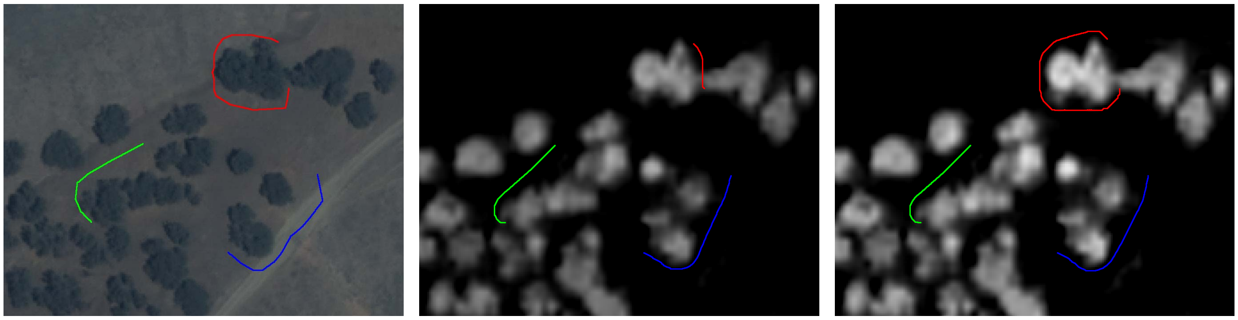
□

By (5.3.2) the similarity between inputs to the projections is negatively correlated to the overlap of the positive and negative visitation counts. When there exists clear differentiation between what features should have their costs increased and decreased, the projection inputs will be similar. As the example and current planned behaviors travel over increasingly similar terrain, the inputs begin to diverge; the contribution of the balanced projection to the current cost function will level out, while that of the unbalanced projection will increase in the direction of the longer path. Finally, in a fully unachievable case, the balanced projection will zero out, while the unbalanced would drive the cost in the direction of the more dominant class. Due to the projection of the functional gradient, these effects hold not only when the terrain in question is identical, but also when it is sufficiently similar as to be indistinguishable to the chosen class of regressor. These effects are observed empirically in Section 5.5. The implementation of this balancing is shown in Algorithm 4.

5.3.2 Noisy Demonstration: Replanning and Corridor Constraints

A balanced regression can help to account for large scale sub-optimality in human demonstration. However, sub-optimality can also occur at a smaller scale. It is unreasonable to ever expect a human to drive or demonstrate the exact perfect path; it is often the case that a plan that travels through neighboring or nearby states would be a slightly better example. In some cases this example noise translates to noise in the cost function; in more extreme cases it can significantly affect performance (Figure 5.4). What is needed is an approach that smoothes out small scale noise in expert demonstration, producing a better training example.

Such a smoothed example can be derived from expert demonstration by redefining the MMP constraint: instead of example behavior being interpreted as the exact optimal behavior, it can be interpreted as a behavior that is spatially near to the optimal path. The exact definition of close



(a) Example Paths (b) Planned Paths (No Replanning) (c) Planned Paths (With Replanning)

Figure 5.4: An example of how noisy demonstration can hurt performance. The red and green example paths in (a) are drawn slightly too close to trees, preventing the cost of the trees from increasing sufficiently to match the red example (b). However, if the paths are allowed to be replanned within a corridor, the red and green path are essentially smoothed, allowing the cost of trees to get sufficiently high (c). On average, the trees achieve three times the cost in (c) as in (b).

depends on the state space; the loss function will always provide at least one possible metric. If the state space is $\mathbb{R}^n$, then Euclidean distance is a natural metric. Therefore, rather than an example defining the exact optimal path, it would define a corridor in which the optimal path exists.

Redefining the original MMP constraint in (5.1.4) in this way (and converting to general as opposed to linear cost functions) yields

$$\begin{aligned} \text{minimize } \ \mathcal{O}[C] = \lambda \text{R}\textsc{eg}[C] \\ \text{subject to the constraint} \\ \sum_{x \in P_*} (C(F_x) - L_e(x)) \ \geq \ \sum_{x \in P_e^*} C(F_x) \\ P_* = \arg\min_P \sum_{x \in P} (C(F_x) - L_e(x)) \\ P_e^* = \arg \min_{P \in \mathcal{N}_e} \sum_{x \ inP} C(F_x) \end{aligned} \tag{5.3.3}$$

Instead of enforcing that P_e is optimal, the new constraint is to enforce that P_e^* is optimal, where P_e^* is the optimal path within some set of paths $\mathcal{N}_e$. The definition of $\mathcal{N}_e$ determines how 'close' is defined. Using the above example of a corridor in a Euclidean space, $\mathcal{N}_e$ would be defined as

$$\mathcal{N}_e = \{P \mid \forall x \in P \ \ \exists y \in P_e \quad s.t. \ ||x - y|| \leq \beta\}$$

with β defining the corridor width. In the general case, this definition can always be rewritten in terms of the loss function

$$\mathcal{N}_e = \{P \mid \forall x \in P \ \ \exists y \in P_e \quad s.t. \ L(x, y) \leq \beta\}$$

It is important to note that the definition of N_e is only dependent on individual states. Therefore,

P_e^* can be found by an optimal planner, simply by only allowing traversal through states that meet the loss threshold β with respect to some state in P_e.

Reformulating (5.3.3) as an optimization problem yields the following objective

$$\begin{aligned} \text{minimize } \mathcal{O}[C] = & \lambda \textsc{Reg}(C) \\ & + \min_{\hat{P}_e \in \mathcal{N}_e} \left[\sum_{x \in \hat{P}_e} C(F_x) \right] \\ & - \min_{\hat{P}} \left[\sum_{x \in \hat{P}} (C(F_x) - L_e(x)) \right] \end{aligned} \tag{5.3.4}$$

The resulting change in the LEARCH algorithm is to carry through the extra minimization to the computation of the visitation counts. That is, at every iteration, a new, smoothed, example is chosen from with $\mathcal{N}_e$; example visitation counts are computed with respect to this path. Combining this new step (example replanning) with weight balancing results in an algorithm known as Robust LEARCH (R-LEARCH). These robust extensions, combined with the dynamic adaptations, are presented in Algorithm 4 as the Dynamic Robust LEARCH (DR-LEARCH) algorithm

It should be noted that as a result of this additional, non-negated min term, the objective is no longer convex. It is certainly possible to produce individual examples where such a smoothing step can result in poor local minima; however, it has been observed empirically that this effect is neutralized when using multiple examples. The experimental performance of this smoothing step is presented in Section 5.5.

When operating with dynamic and partially unknown perceptual data, this replanning step provides another important side effect. Rewriting (5.2.3) with this additional min term yields

$$\begin{aligned} \text{minimize } \mathcal{O}[C] = \ & \lambda \textsc{Reg}(C) \\ + \ \sum_t \min_{\hat{P}_e^t \in \mathcal{N}_e^t} & \left(\sum_{x \in \hat{P}_e^t \cap \mathcal{V}^t} C(F_x^t) \ + \sum_{x \in \hat{P}_e^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right) \\ - \ \sum_t \min_{\hat{P}^t} & \left(\sum_{x \in \hat{P}^t \cap \mathcal{V}^t} (C(F_x^t) - L_e^t(x)) \ + \sum_{x \in \hat{P}^t \cap \bar{\mathcal{V}}^t} C_{\bar{\mathcal{V}}}(x) \right) \end{aligned} \tag{5.3.5}$$

Where $\mathcal{N}_e^t$ is the set of paths near P_e^t. However, as before it should be noted that the $C_{\bar{\mathcal{V}}}$ terms will have no affect on the functional gradient. Therefore, the definition of $\mathcal{N}_e^t$ does not need to consider states in $\bar{\mathcal{V}}$. This yields the following general definition of $\mathcal{N}_e^t$

$$\mathcal{N}_e^t = \{ P^t \mid \forall x \in P^t \bigcap \mathcal{V}^t \ \exists y \in P_e^t \bigcap \mathcal{V}^t \quad s.t. \ L(x, y) \leq \beta \}$$

the result is that $\mathcal{N}_e^t$ only defines closeness over $\mathcal{V}^t$. Behavior outside of $\mathcal{V}^t$ does not directly affect the gradient, but does affect the objective value (the difference in cost between the current (replanned) example and planned behavior). Therefore, by performing a replanning step (even with $\beta = 0$), example behavior can be made consistent with $C_{\bar{\mathcal{V}}}$ without compromising its effectiveness as an example within $\mathcal{V}^t$. This notion of consistency proves to have meaningful value.

Algorithm 4: The Dynamic Robust LEARCH algorithm

```
Inputs : Example Behaviors $P_e^1, P_e^2, ..., P_e^n$, Sensor Histories $\mathcal{H}^1, \mathcal{H}^2, ..., \mathcal{H}^n$, Cost Map
         $C_{\bar{\mathcal{V}}}$, Corridor width $\beta$
$C_0 = 1$;
foreach $P_e^i$ do
    for $\tau$ =firstTime($P_e^i$): $\Delta\tau$ :lastTime($P_e^i$) do
        $P_e^{\tau,i}$ = extractPathSegment($P_e^i$, $\tau$,lastTime($P_e^i$));
        [$\mathcal{F}^{\tau,i}$, $\mathcal{V}^{\tau,i}$] = simulatePerception($\mathcal{H}^i$,firstTime($P_e^i$),$\tau$);
for $j = 1...K$ do
    $U +_= U_- = \vec{0}$;
    foreach $P_e^{t,i}$ do
        $\mathcal{M}^{t,i}$ = buildCostmap($C_{i-1}$, $\mathcal{F}^{t,i}$,$\mathcal{V}^{t,i}$,$C_{\bar{\mathcal{V}}}$);
        $P_*^{t,i}$ = planLossAugmentedPath(start($P_e^{t,i}$),goal($P_e^{t,i}$),$\mathcal{M}^{t,i}$);
        $\mathcal{M}_{\beta,\mathcal{V}^{t,i}}^{t,i}$ = buildCorridorRestrictedCostmap($\mathcal{M}^{t,i}$,$\beta$,$\mathcal{V}^{t,i}$);
        $P_{e*}^{t,i}$ = replanExample(start($P_e^{t,i}$),goal($P_e^{t,i}$),$\mathcal{M}_{\beta,\mathcal{V}^{t,i}}^{t,i}$,);
        foreach $x \in P_{e*}^{t,i} \bigcap \mathcal{V}^{t,i}$ do
            $U_-(\mathcal{F}_x^{t,i}) = U_-(\mathcal{F}_x^{t,i}) + 1$;
        foreach $x \in P_*^{t,i} \bigcap \mathcal{V}^{t,i}$ do
            $U_+(\mathcal{F}_x^{t,i}) = U_+(\mathcal{F}_x^{t,i}) + 1$;
    $T_f = T_o = T_w = \emptyset$;
    $U = U_+ - U_-$;
    $N_+ = N_- = 0$;
    foreach $\mathcal{F}_x^{t,i}$ such that $U(\mathcal{F}_x^{t,i}) \neq 0$ do
        $T_f = T_f \bigcup \mathcal{F}_x^{t,i}$;
        $T_o = T_o \bigcup$ sgn($U(\mathcal{F}_x^{t,i})$);
        $T_w = T_w \bigcup |U(\mathcal{F}_x^{t,i})|$;
        if sgn($U(\mathcal{F}_x^{t,i})$) $> 0$ then $N_+ = N_+ + 1$ else $N_- = N_- + 1$;
    foreach $(t_o, t_w) \in (T_o, T_w)$ do
        if $t_o > 0$ then $t_w = t_w/N_+$ else $t_w = t_w/N_-$;
    $R_j$ = trainWeightedRegressor($T_f$,$T_o$,$T_w$);
    $C_j = C_{j-1} * e^{\eta_j R_j}$;
return $C_K$
```

5.3.3 Filtering for Inconsistent Examples

One fundamental issue with expert demonstration is consistency. A human demonstrator may act approximately according to one metric during one example, and a slightly different metric during another example. While each individual example may be near-optimal with respect to some metric, the two examples together may be inconsistent; that is, there is no consistent cost function that would define both demonstrations as optimal.

The possibility of an expert interpreting unknown terrain in a different manner is a potentially large source of inconsistency. This is especially true when attempting to learn an online cost func-

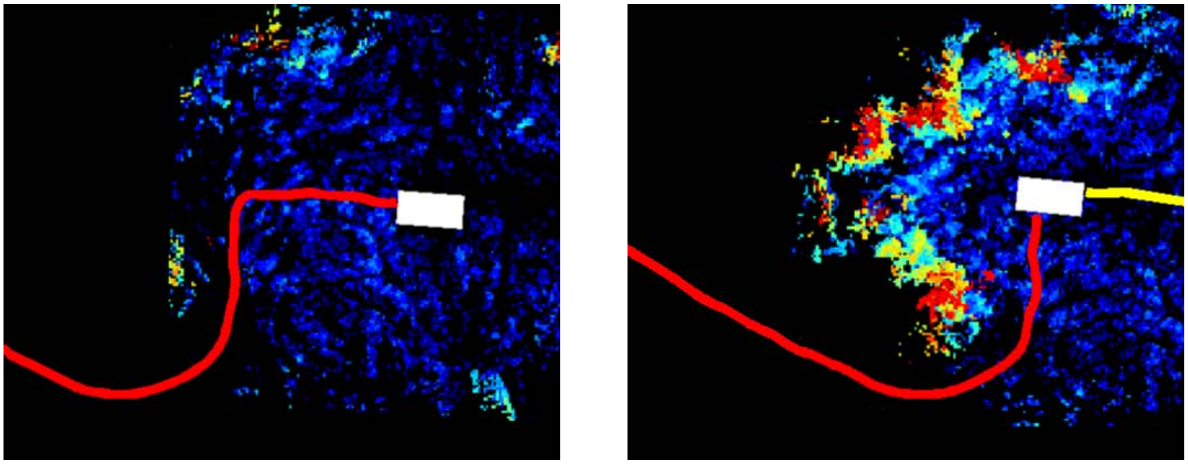


Figure 5.5: Recorded example behavior from time t (left) and $t + 1$ (right), overlayed on a single perceptual feature (obstacle height). Future behavior is inconsistent at time t, but makes sense at time $t + 1$ given additional perceptual data.

tion, as it is very likely that the demonstrator will have implicit prior knowledge of the environment that is unavailable to the perception system. However, by always performing a replanning step as previously discussed, example demonstration can be made consistent with the robot's interpretation of the environment in unobserved regions.

With consistency in unobserved regions accounted for, there remain four primary sources of inconsistent demonstration

- Inconsistency between multiple experts
- Expert error (poor demonstration)
- Inconsistency between an expert's and the robot's perception in observed regions
- A mismatch between an expert's planned and actual behavior

This last issue was alluded to in Section 5.2: while an expert example should consist of an expert's *plan* at time t from the current state to the goal, what is recorded is the expert's *behavior* from time t to the goal. Figure 5.5 provides a simple example of this mismatch: at time t, the expert likely planned to drive straight, but was forced to replan at time $t + 1$ when the cul-de-sac was observed. This breaks the assumption that the expert behavior from time t onward matches the expert plan; the result is that the discretized example at time t is inconsistent with other example timesteps.

However, the very inconsistency of such timesteps provides a basis for their filtering and removal. Specifically, it can be assumed that a human expert will plan in a fairly consistent manner during a single example traverse[7]. If the behavior from a single timestep or small set of timesteps is inconsistent with the demonstrated behavior at other timesteps, then it is safe to assume that this small set of timesteps does not demonstrate correct behavior, and can be filtered out and removed as training examples. This does not significantly affect the amount of training data required to train

[7] If this assumption does not hold, then the very idea of learning from said expert's demonstration is flawed

a full system, as by definition an inconsistent timestep was unlikely to provide a useful example. This approach is similar to that of [237, 238], which both investigate the identification and removal of noisy or incorrect class labels when using boosting methods.

Inconsistency can be quantitatively defined by observing each timestep's contribution to the objective functional (its slack penalty). In (5.1.5) this penalty is explicitly defined as a a measurement of by how much a constraint remains violated. If the penalty at a single of an example behavior timestep is a statistical outlier from the distribution of slack penalties at all other timesteps, it indicates that single timestep implies constraints that remain violated far more so than others. That is, the constraints at an outlier timestep are inconsistent with those implied the rest of a demonstration.

Therefore, the following filtering heuristic is proposed as a pre-processing step. First, attempt to learn a cost function over all timesteps of a single example behavior and identify timesteps whose penalties are statistical outliers. During this step, a more complex hypothesis space of cost functions should be used than is intended for the final cost function (i.e use more complex regressors). As these outlier timesteps are inconsistent within an overly complex hypothesis space, there is evidence that the inconsistency is in the example itself, and not for lack of expressiveness in the cost function. Therefore, these timesteps should be removed. This process can be repeated for each example behavior, with only remaining timesteps used in the final training.

Aside from filtering out inconsistency due to plan/behavior mismatch, this approach will also filter timesteps due to other sources of inconsistency. This is beneficial, as long as the timesteps truly are inconsistent. However, the possibility always remains that the example itself was correct; it may only appear inconsistent due to the fidelity of perception or planning. In this case, filtering is still beneficial, as the examples would not have been learnable (with the current set of perceptual features and the current planning system); instead, the small subset of filtered examples can be examined by a system expert, who may then identify a necessary additional component level capability. Experimental results of this filtering approach are presented in Section 5.5.

5.4 Application to Mobile Robotic Systems

Before either R-LEARCH or DR-LEARCH can be applied to the task of learning a terrain cost function for a mobile robotic system, there are still some practical considerations to address. It is important to remember the specific task for which LEARCH and its variants are intended: they are designed to select a cost function from a defined hypothesis space $\mathcal{C}$, such that expert demonstration is recreated when the cost function is applied to the *specific* perception and planning systems for which it was trained. There are several hidden challenges in that statement, such as defining $\mathcal{C}$, and ensuring LEARCH is producing a cost function for the correct planning system.

5.4.1 Selecting a Cost Function Hypothesis Space

The cost function hypothesis space $\mathcal{C}$ is implicitly defined by the regressor space $\mathcal{R}$. In turn, $\mathcal{R}$ is defined by design choices relating to the family and allowable complexity of regressors. For example, if single layer neural networks with at most H hidden are chosen as the class of regressors, than $\mathcal{C}$ consists of all cost functions that are a weighted sum of such networks. In this way, cost functions of almost arbitrary complexity can be allowed.

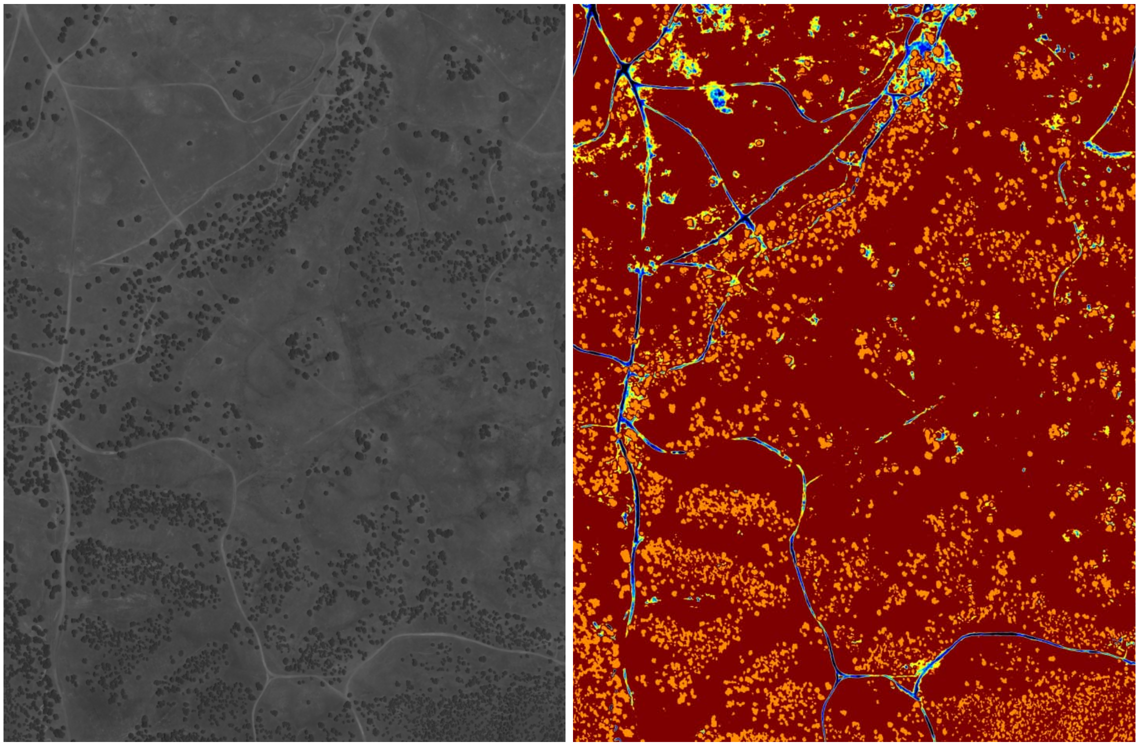


Figure 5.6: Example of a new feature (right) learned automatically from panchromatic imagery (left) using only expert demonstration (there are no explicit class labels).

However, as with most machine learning techniques, there is a design tradeoff between expressiveness and generalization. Complex regressors are capable of expressing complex costing rules, but are more prone to overfitting. In contrast, simpler regressors may generalize better, but are also limited in the costing rules they can express. This tradeoff must be effectively balanced to ensure both sufficient expressiveness and generalization. Fortunately, as regression is a core machine learning task, there are well developed and understood approaches for achieving this balance. In practice, validation with an independent holdout set of demonstrated examples can quantify this tradeoff. This allows a range of regressor types and complexities to be automatically evaluated; the one with the best holdout performance can be selected with minimal additional human interaction.

Another issue with respect to the definition of $\mathcal{C}$ is computational cost. In a scenario where perceptual features are static, this concern is not as important, as cost evaluations are only performed once. However, in an online and dynamic setting, cost evaluations are performed continuously; the computational complexity may be a much larger issues. As the final learned cost function is a weighted combination of K regressors, computational complexity of the final cost function is linear in K. Again, this creates a design tradeoff; limiting K will limit the computation cost per evaluation, but will also limit the accuracy of the cost function (as the final steps of a gradient descent operation fine tune the solution).

One solution to this problem would be to relearn a cost function after LEARCH completes; that is, learn a single regressor R_p to minimize $(R_p(F) - C_K(F))^2$ for a large sample of $F \in \mathcal{F}$. Unfortunately, this approach would create the need for additional validation of R_p as its complexity may have to be continually modified to minimize training or test set error. Additionally, there would be no guarantee that R_p would reproduce similar behavior (as small changes in costs can have large effects, and vice versa). Another solution would be to define $\mathcal{R}$ as the space of linear functions. Since the weighted sum of linear functions is another linear function, using this definition of $\mathcal{R}$ would result in the final complexity of C_K being constant with respect to K. However using linear regressors with LEARCH results in almost the same solution as would be produced by the linear MMP algorithm (but not identical due to exponentiated functional gradient descent). As a fundamental advantage of the LEARCH approach was to allow non-linear cost functions, this would seem to imply an undesirable but necessary tradeoff.

The addition of a feature learning phase [225, 229, 230] to LEARCH can potentially solve this problem. During most learning iterations, linear regressors are used. However, when it is detected that the objective error is no longer decreasing, a single, simple non-linear regression is performed. This single, non-linear step would better discriminate between terrains that are difficult for a purely linear regressor. However, rather than add this new regressor directly into the cost function, it is instead treated as a new feature. In this way, future iterations can fine tune the weight on this feature. If the non-linear step is performed with a rule based regressor (such as a regression tree), then this approach begins to look similar to the way in which cost functions are often hand tuned: simple linear functions handle the general cases, with simple sets of rules to handle difficult special cases. LEARCH with a feature learning phase is presented in Algorithm 5 (For clarity, it is shown only for the static case; extension to DR-LEARCH is straightforward). It should also be noted that once a new feature is learned, it does not need to be computed for the entire feature space a priori; this can be done opportunistically for efficiency.

An additional advantage of this approach is that new learned features can be used as a guide in the development of new engineered feature extractors. Figure 5.6 provides a simple example. In this context, LEARCH attempted to learn a linear cost function from only a single feature (the greyscale value of each pixel). Such an approach is doomed to failure, as intensity is not linearly related to cost. Therefore, a single new feature was learned using a regression tree with 8 leaf nodes. This new feature strongly disambiguates roads and trails from surrounding terrain. Not only does this improve the learned cost function, it can also be taken as an indication that an explicit road or trail extractor would be useful, and worth devoting engineering resources to.

5.4.2 Planner Interpretation of Cost Maps

A common approach in mobile robotics is to treat the environment as a 2.5D space; this allows the state space $\mathcal{S}$ for high level path planning to be $\mathbb{R}^2$. This results in terrain costs defined over a a discretized 2D grid. However, different planning systems may interpret a 2D cost grid differently. Since the goal is to learn a cost function to recreate behavior with a specific planning system, these details must be taken into account to learn the cost function correctly.

Perhaps the simplest cost-aware planner one might use for a mobile robot would be 4-connected A* (or other grid planner). Such a planning system would incur a cost of $C(x)$ whenever a cell x was traversed. Now consider the case of an 8-connected A*. Many 8-connected implementations treat traversing a cell diagonally as higher cost than traversing the same cell axis-aligned; this allows the planner to take into account the extra distance traversed through a grid. This usually takes the form of incurring a cost of $C(x)$ when traversing the cell axis-aligned, and a cost of $\sqrt{2}C(x)$ when traversing the cell diagonally.

Since cells traversed diagonally incur more cost, this must be taken into account by LEARCH

Algorithm 5: The linear LEARCH algorithm with a feature learning phase

```
Inputs : Example Paths $P_e^1, P_e^2, ..., P_e^n$, Feature Map $\mathcal{F}$
$C_0 = 1$;
for $j = 1...K$ do
    $\mathcal{M} =$ buildCostmap($C_{j-1}, \mathcal{F}$);
    $U_+ = U_- = \vec{0}$;
    foreach $P_e^i$ do
        $P_*^i =$ planLossAugmentedPath(start($P_e$),goal($P_e$),$\mathcal{M}$);
        foreach $x \in P_e^i$ do
            $U_-(\mathcal{F}_x) = U_-(\mathcal{F}_x) + 1$;
        foreach $x \in P_*^i$ do
            $U_+(\mathcal{F}_x) = U_+(\mathcal{F}_x) + 1$;
    $T_f = T_o = T_w = \emptyset$;
    $U = U_+ - U_-$;
    foreach $\mathcal{F}_x$ such that $U(\mathcal{F}_x) \neq 0$ do
        $T_f \ = T_f \bigcup \mathcal{F}_x$;
        $T_o \ = T_o \bigcup$ sgn($U(\mathcal{F}_x)$);
        $T_w = T_w \bigcup |U(\mathcal{F}_x)|$;
    $R_j =$ trainWeightedLinearRegressor($T_f, T_o, T_w$);
    $C_j = C_{j-1} * e^{\eta_j R_j}$;
    if !hasPerformanceImproved($C_j, C_{j-1}, P_e^1, ..., P_e^n$) then
        $R_b =$ trainWeightedNonlinearRegressor($T_f, T_o, T_w$);
        foreach $\mathcal{F}_x \in \mathcal{F}$ do
            $\mathcal{F}_x =$concat($\mathcal{F}_x, R_b(\mathcal{F}_x)$);
return $C_K$
```

when computing the projection of the functional gradient. With respect to the final cost of a path, a cell traversed diagonally will have a $\sqrt{2}$ greater affect than one traversed axis-aligned; therefore it is $\sqrt{2}$ times more important to get the sign right on the projection. The solution is to increment the visitation count of a cell by 1 when traversing it axis-aligned, and by $\sqrt{2}$ when traversing diagonally. In the general case, a planner may incur a cost of $dC(x)$ when traversing a distance d through cell x; the visitation count of state x should then be incremented by d. Examples of planners that will plan continuously through $\mathbb{R}^2$, even when costs are discrete, include the Field D* [239] algorithm used in [95, 101, 102] and the hybrid A* used in [37].

Another issue that must be considered is that of configuration space expansion. Motion planners for mobile robotic systems often apply a c-space expansion to an input cost map before generating a plan, in order to account for the physical dimensions of the robot. The result is that the cost the planner assigns for traversing distance d through state x is no longer $dC(x)$, but rather something along the lines of

$$d \sum_{y \in \mathcal{N}} W(x, y) C(y) \tag{5.4.1}$$

where $\mathcal{N}$ is a set of states sufficiently close to x, and W is a weighting function. Common choices for W include a constant, or a linear falloff based on $||x - y||$. As before, this weighting must be captured by the visitation counts: if distance d is traversed through cell x, then all cells $y \in \mathcal{N}$ must have there visitation counts incremented by $dW(x, y)$. A further complication arises if W depends not only on the locations of x and y, but also their (or other states) cost values. For instance, if a c-space expansion defined the cost of traversing d through state x as

$$d \max_{y \in \mathcal{N}} C(y)$$

then only the state y with the maximum cost in $\mathcal{N}$ should have its visitation count incremented by d; all other states would not affect the planner perceived cost of traversing x under the current C. Unlike (5.4.1) this form results in non-convexity in the LEARCH algorithm (in addition to non-convexity from replanning), as different initial cost functions C_0 may produce significantly different results.

5.4.3 Planners with Motion Constraints

Even though costs may only be defined over a 2D grid, many motion planning systems for mobile robots still consider the kinematic and dynamic constraints of the vehicle. A common architecture for mobile robot planning systems is to utilize a hierarchy of planners, ranging for long-range low resolution planners to short-range high resolution planners. The simplest form of this architecture utilizes a long-range, unconstrained 'global' planner, and a short-range, kinematically or dynamically constrained 'local' planner [95, 101, 102]. Usually, a local planner does not plan all the way to the goal; instead it produces a set of feasible short-range actions, and utilizes the global planner to produce a path from the end of the action to the goal. Local planner generated plans are not actually followed to completion, but instead are replanned at a high rate. In this way, the local planner is not actually optimal, but instead performs something akin to a greedy search. More detailed discussion of such receding horizon planning systems is delayed until Section 6.2.1.

If DR-LEARCH is to be used to learn costs for an onboard perception system, it is important that they be learned with respect to the planning system that will be used onboard the robot. If a hybrid architecture is used, DR-LEARCH must be implemented using the same configuration. It is important that the decisions the planner is tasked with making during training are of the same form that will be required during operation. For example, in the case of a greedy local planning system, P_*^t at each iteration should be the direct output of the planning system, not the concatenation of multiple planning cycles (even though this is how the robot's actual behavior is generated online). This is necessary for blame and credit assignment; if at time t the expert swerved to avoid an obstacle, then not only must the cost of the obstacle be sufficiently high, but the cost of the terrain swerved over must be sufficiently low. Even if subsequent actions reduce the distance traveled during the swerve, the planner must be willing to perform a larger turn at time t to begin that maneuver.

Figure 5.7 provides a simplified scenario to demonstrate why this is so. In this example, a local/global planning system is imagined that can only consider 3 possible local actions for avoiding an obstacle: straight, hard left, or hard right. From the endpoint of each action, the global planner plans straight to the goal. In contrast, a path around the obstacle from the unconstrained global planner is also shown. If costs were trained with respect to the unconstrained planner, DR-

LEARCH would be satisfied with the cost on the obstacle when it is sufficiently high to make up for the extra distance $|P_g| - |P_c|$ necessary to go around the obstacle rather than over it. That is, $|P_g| < |P_c| + O$, where O is the cost of the obstacle; O only needs to equal $|P_g| - |P_c| + \epsilon$ for $|P_g| < |P_c| + O$. However, the constrained planner cannot consider P_g. It is therefore possible for $|P_l|, |P_r| > |P_c| + O > |P_g|$; P_c remains the cheapest constrained option in this case. The result would be that the constrained local planner would still choose to drive over the obstacle. By extension to more realistic planning systems, DR-LEARCH must utilize the lowest level (constrained) planner when learning a cost function under such a hierarchy.

Unfortunately, there are also problems with training directly for constrained planners. As a constrained planner only considers a discrete set of feasible actions, it is often the case that no series of actions will ever exactly match the expert example. In this case, it is unclear when DR-LEARCH has learned sufficiently to terminate. Terminating when the example path is lower cost than the planned path will not suffice; in Figure 5.7 this could result in $C(P_e) < C(P_c) < C(P_l), C(P_r)$ (the colliding action would still be preferred). Running until the cost function itself converges would therefore appear necessary. Unfortunately, this has its own side effects. Once $C(P_c) > C(P_l), C(P_r)$, DR-LEARCH will try to raise the cost along P_l or P_r. If the chosen regressor is powerful enough to differentiate between the terrain under P_l or P_r and that under P_e, it will raise those costs without proper cause. The end result is a potential addition of noise to the final cost function, and poorer generalization. The degree of this noise depends on both the resolution of the constrained planner and the expressiveness of the regressor.

What would be ideal is if for every instant in time along an example, it was known which action from amongst a constrained planner's action set the expert would choose, as well as the unconstrained path that an expert would traverse from the end of the action. Collecting this information during demonstration by an expert would be extremely tedious, requiring an expert action selection and path generation at every planning cycle. However, a boostrapping approach can suffice to provide the latter half of this requirement. Since LEARCH and its variants can be applied directly to the unconstrained planner, and LEARCH seeks a cost function that reproduces expert example behavior, a cost function learned for an unconstrained planner can be used to generate examples for a constrained one. That is, the path an expert would take from the end of an action can be predicted using the cost function learned under such unconstrained conditions [240], and used to generate feasible examples for the unconstrained planner. Now, the problem is simply to estimate which path in the constrained planners action set an expert would choose at each discrete timestep. The boostrapped example path is then the concatenation of this action and the planned path under the cost function learned for the unconstrained planner. For example, in Figure 5.7 the example path would be transformed from P_e into P_l.

The easiest solution to this new problem is simply to record what action an expert executed at each instant during demonstration, and somehow project this onto a constrained planner's action set (this approach is more concretely defined and applied in Chapter 6). However, this assumes that the expert was paying attention to what actions were being executed as well as over what terrain the robot was traversing. If this is not the case, then there is no reason to try and mimic the expert's actions, only where the expert traversed. While certainly not desirable, such conditions can occur for difficult to control robotic systems (such as the Crusher vehicle used in these experiments). Under such circumstances, the following heuristic is proposed. First, a cost function is learned according to the unconstrained planner (as must be done anyway for boostrapping). As described above, such a cost function will generally underestimate the cost necessary for the equivalent

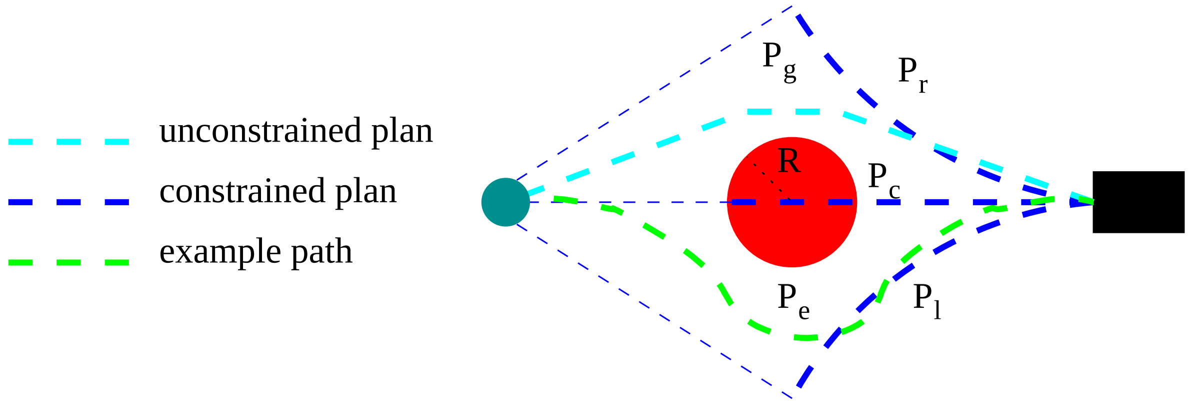


Figure 5.7: A simplified scenario where a kinematically constrained planner has only 3 possible actions

constrained planner. Therefore, each potential constrained plan is scored by its *average* cost instead of total cost. A plan with low average cost can not be said to be optimal, but it at least traverses desirable (low cost) terrain. An additional penalty based on path length is then added to bias action scores towards those that make progress towards the goal[8]. After scoring each action, that with the lowest score is used as the new example action. This form of heuristic would be applicable whenever an expert demonstration only provides information on what a robot should do, but not specifically how to do it.

5.5 Experimental Results

Learning preference models by demonstration in the MMP framework was applied to the Crusher autonomous system (Figure 5.8). The Crusher vehicle is capable of traversing rough, complex, and unstructured terrain; as such understanding the relative benefits and tradeoffs of various terrain type is of paramount importance to its autonomous operation. On Crusher, terrain data comes from two primary sources. The first is prior, static, overhead data sources (satellite or aerial imagery, aerial LiDAR, etc.). Overhead data is processed via an set of engineered feature extractors into a set of feature maps, which are then mapped into a single, static costmap for an entire area of operation. The second source of terrain data comes from the onboard perception system. The onboard perception system processes local data from onboard camera images and LiDAR into a dynamic stream of features. At a high data rate, these features are continuously mapped to costs over a local area.

Costs from both sources are continuously fused into a single consistent costmap, which is then passed to Crusher's motion planning system. Fusing prior and onboard perceptual data at the cost level allows for Crusher to continuously plan a path all the way from its current position to the goal. Due to the dynamic nature of this cost data, the Field D* algorithm is utilized [239]. In order to determine feasible local motion commands for Crusher, a variant of the RANGER system [59, 95] is applied, utilizing the Field D* plan for global guidance (See Section 6.2.1 for more details). The

[8] the weight of this penalty can be automatically tuned by optimizing performance on a validation set, without any hand tuning

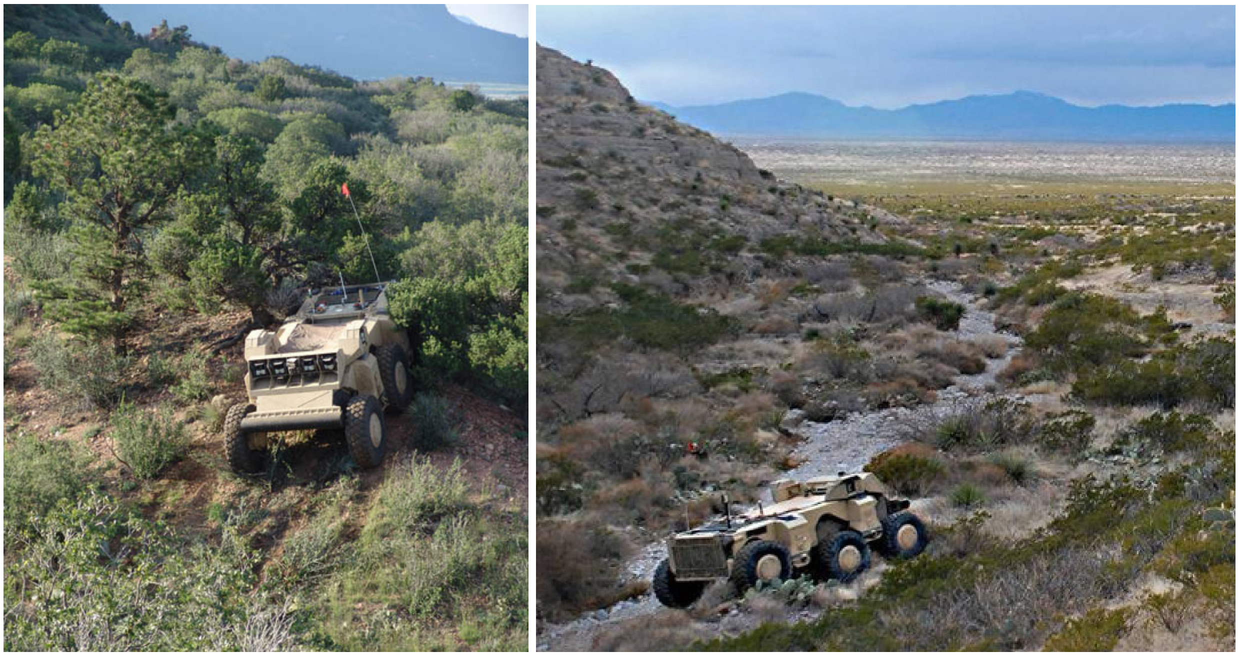


Figure 5.8: The Crusher autonomous mobile platform used in work to date. Crusher is capable of cross-country traverse through rough, complex, and unstructured terrain

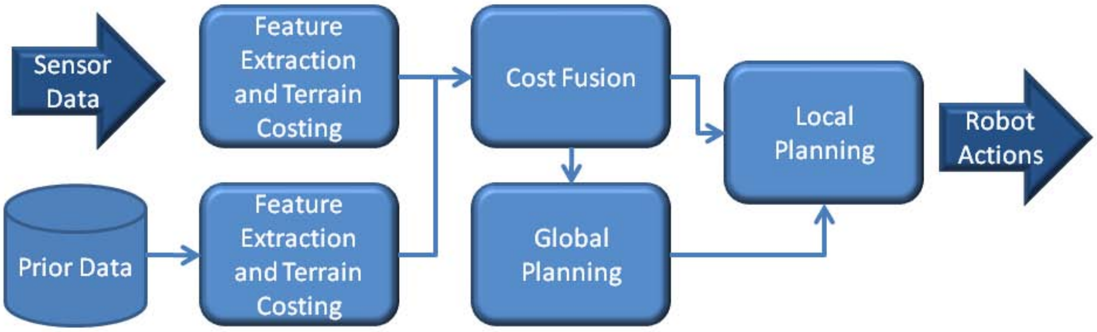
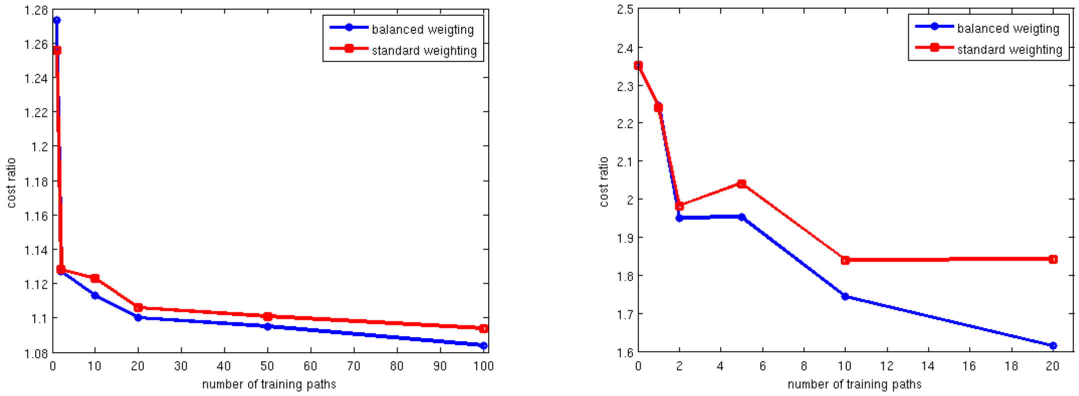

Sensor Data
Feature Extraction and Terrain Costing
Cost Fusion
Local Planning
Robot Actions
Prior Data
Feature Extraction and Terrain Costing
Global Planning

Figure 5.9: A high level block diagram of the Crusher Autonomy system

architecture of Crusher's autonomy system is shown is Figure 5.9.

Early in Crusher's development, the task of interpreting both static and dynamic perceptual data into costs was accomplished via hand tuning, and was a source of frustration. This led to the application of LEARCH to constructing costing models. This was first applied in the static perceptual case (overhead data) utilizing the Field D* planner, and was next applied to the dynamic case (onboard perceptual data) utilizing local/global hierarchical planning system. The remainder of this section describes these experiments, along with offline results from each task.

Two metrics are used for evaluating offline performance. The first is the average loss along a path. As the loss function is constructed to encode the similarity between two paths, the loss between an example path and the corresponding planned path (over the interval [0,1]) is a measure of how accurately expert behavior has been reproduced. A second measure is the cost ratio, defined as the cost of an example path divided by the cost of the corresponding planned path. As this ratio



(a) Simulated Examples (b) Expert Drawn Examples

Figure 5.10: Learning simulated and expert drawn example paths. Test set performance is shown as a function of number of input paths

approaches 1, it indicates the current cost function hypothesis is approaching consistency with the expert demonstration. The cost ratio as opposed to cost difference is used to account for the effect of scaling on the cost function (with the cost difference, simply scaling the costs closer to zero would improve the metric, without improving the relative performance).

5.5.1 Learning to Interpret Overhead Data

In order to verify LEARCH and R-LEARCH under ideal conditions, tests were first run on simulated examples. A known (arbitrary) cost function was used to generate a cost map over a single environment from its overhead features; this cost map was used to produce paths between random waypoints. Different numbers of these paths were then used as input for LEARCH, and the performance measured on an a large independent validation set of paths (generated in the same manner.)

Figure 5.10(a) shows the results using both the balanced and standard weighting schemes (Section 5.3.1). As the number of training paths is increased, the test set performance continues to improve. Each input path further constrains the space of possible cost functions, bringing the learned function closer to the desired one. However, there are diminishing returns as additional paths overlap to some degree in their constraints (see Section 7 for more on this effect). Finally, the performance of the balanced and standard weighting schemes is similar. Since all paths for this experiment were generated by a planner, they are by definition optimal under some metric, and therefore both achievable and consistent with each other.

Next, experiments were performed with expert examples (both training and validation) drawn on top of overhead data maps. Figure 5.10(b) shows the results of an experiment of the same form as that performed with simulated examples. Again, the validation set cost ratio decreases as the number of training examples increases. However, with real examples there is a significant difference between the two weighting schemes; the balanced weighting scheme achieved significantly better performance. This demonstrates both how human demonstration is naturally noisy and imperfect, and how R-LEARCH is robust to this fact through a balanced regression.

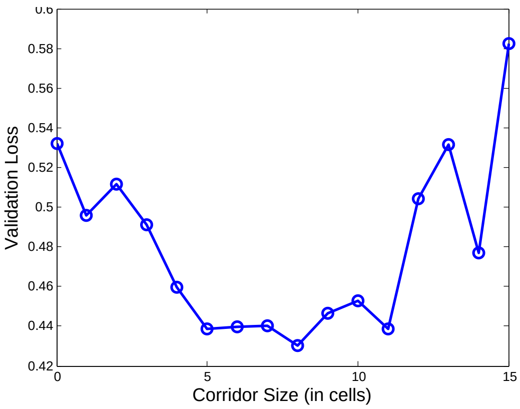


Figure 5.11: Validation loss as a function of the corridor size. Using corridor constraints improves performance as long as the corridor is not too large

Another series of experiments were performed to determine the effect of performing replanning with corridor constraints (Section 5.3.2). For these experiments, the performance of learning was measured with validation loss, to indicate how well the planned paths matched the examples. When measuring loss on the validation set, no replanning was performed. Therefore, in order to provide a smoother metric, the specific loss function used was a radial basis function between states on the current path P_* and the closest state on the example P_e, with a scale parameter σ^2

$$L(P_*, P_e) = \frac{1}{|P_*|} \sum_{x \in P_*} [1 - \exp\big(\min_{x_i \in P_e} [\|x - x_i\|^2]/\sigma^2\big)] \tag{5.5.1}$$

Using a loss function of this form provides a more analog metric than a Hamming style loss as previously described. Figure 5.11 shows the results on the validation set as a function of the corridor size (in cells). Small corridors provide an improvement over no corridor, demonstrating how small scale smoothing can improve generalization. However, as the corridor gets too large, this improvement disappears; large corridors essentially over-smooth the examples and begin to miss critical information.

Finally, experiments were performed in order to compare the offline performance of learned costmaps with hand tuned ones. A cost map was trained off of satellite imagery for an approximately 60 km^2 size environment. A hand tuned costmap had been previously produced for this same test site to support Crusher operations. This map was produced by performing a supervised classification of the imagery, and then manually determining a cost for each class [105]. A subset of both maps is shown in Figure 5.13. The two maps were compared using a validation set of paths generated by a Crusher team member not directly involved in the development of overhead costing. The average validation loss using the learned map was 23% less than the hand tuned map (Figure 5.12), thus demonstrating superior generalization of the learned approach.

Online validation of the learned costmaps was also achieved during Crusher field testing. These field tests consisted of Crusher autonomously navigating a series of courses, with each course defined as a set of widely spaced waypoints. Courses ranged in length up to 20 km, with waypoint spacing on the order of 200 to 1000 m. These tests took place at numerous locations across the continental U.S., each with highly varying local terrain characteristics, and sizes ranging from tens

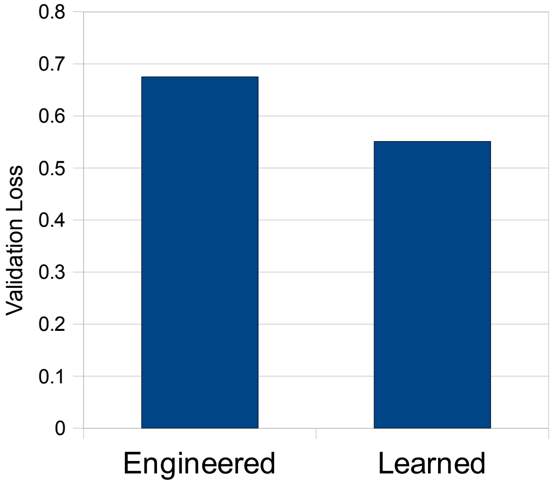


Figure 5.12: Performance comparison between a learned and hand tuned prior costmap. The learned map produced behavior that better matched an independent validation set of examples.

Experiment	Total Net Distance(km)	Avg. Speed(m/s)	Total Cost Incurred	Max Cost Incurred
Experiment 1 Learned	6.63	2.59	11108	23.6
Experiment 1 Hand Tuned	6.49	2.38	14385	264.5
Experiment 2 Learned	6.01	2.32	17942	100.2
Experiment 2 Hand Tuned	5.81	2.23	21220	517.9
Experiment 2 No Prior	6.19	1.65	26693	224.9

Table 5.1: Results of experiments comparing learned to hand tuned prior maps. Indicated costs are from the vehicle's onboard perception system.

to hundreds of square kilometers.

During field testing in 2005 and 2006, prior maps were primarily generated as described in [105]. An initial implementation of the LEARCH algorithm was also demonstrated during smaller tests in 2006. During 2007 and 2008, R-LEARCH became the default approach for producing cost maps from prior data. Overall, R-LEARCH maps were used during more than 600 km of sponsor monitored autonomous traverse, plus hundreds of kilometers more of additional field testing. This demonstrated that a learned cost function was sufficient for use online a complex robotic system.

In addition, two direct online comparisons were performed. These two tests were performed several months apart, at different test sites. During each experiment, the same course was run twice times, varying only the prior cost map given to the vehicle between runs. The purpose of these experiments was to demonstrate that learning a cost function not only generalized better with respect

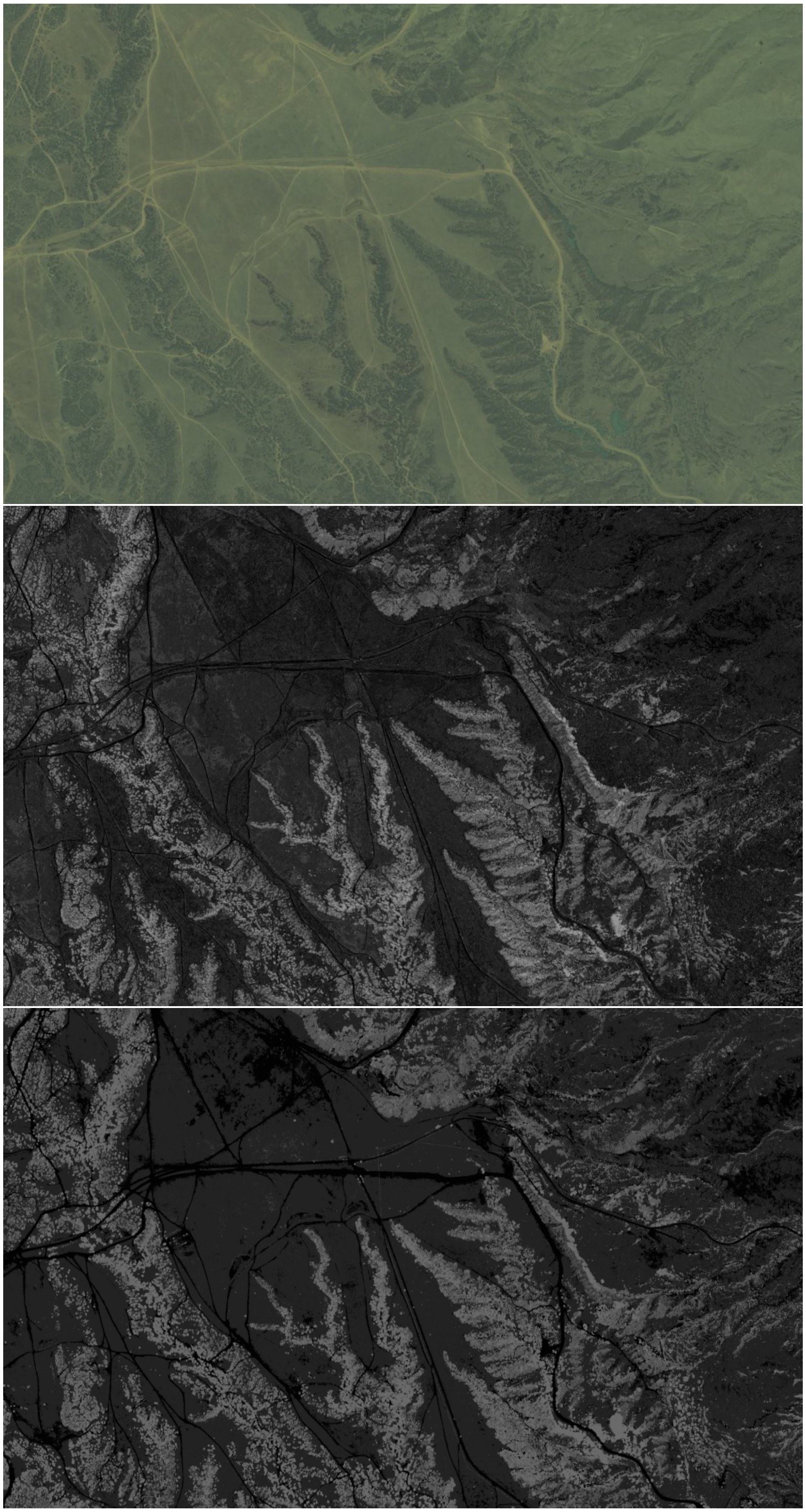


Figure 5.13: A 10 km^2 section of a Crusher test site. From top to bottom: Quickbird imagery, Learned Cost, and Hand Tuned Cost

to initial route planning, but also with respect to dynamic replanning online. Each run was scored according to the total cost incurred by the vehicle according to its onboard perception system. At the time of these experiments, the perception system made use of a manually constructed and tuned cost function. However, this function was shown through numerous experiments [101, 102] to result in a high level of autonomous performance; therefore it is a valid metric for scoring the safety of a single autonomous run.

The results of these experiments are shown in Table 5.1. In both experiments, the vehicle traveled farther to complete the same course using learned prior data, and yet incurred less total (online) cost. Over both experiments, with each waypoint to waypoint section considered an independent trial, the improvement in average cost and average speed[9] is statistically significant at the 5% and 10% levels, respectively. This indicates that the terrain the vehicle traversed was on average safer when using the learned prior map, according to its own onboard perception system. This normalization by distance traveled is necessary because the learned prior and hand tuned perception cost functions do not necessarily agree in what they consider unit cost. Additionally, the maximum cost incurred at any point along an experiment is also provided; for both terrains, the maximum is significantly lower when using the learned prior data. The course for Experiment 2 was also run without any prior data; the results are presented for comparison to this scenario [105].

In addition to improving vehicle performance, using learned cost functions also reduced the necessary amount of human interaction. When preparing for a Crusher test using hand tuned costmaps, performing a supervised classification and tuning the cost function would take on the order of 1-2 days. In contrast, when using learned costmaps drawing example paths would require on the order of 1-2 hours[10]. In a timed head-to-head experiment on a small 2 km^2 test site, producing a supervised classification required 40 minutes of expert involvement, and tuning a cost function required an additional 20 minutes. In contrast, producing example paths required only 12 minutes. On this same experiment, the learned costmap had a validation loss of 0.43, compared to 0.56 for the hand tuned map. This demonstrates that the learned approach results in superior performance, with less human interaction time (Figure 5.14).

An additional test was performed in which the same training set of example paths was used to learn a cost function only from the results of the supervised classification; in this case the learned map had a validation loss of 0.52. This demonstrates two important points. The first is that even when the problem of learning a cost function was reduced to solely a low dimensional parameter tuning problem (in this case 5 dimensions), the automated approach was able to perform better than manual tuning, and with less human interaction. The second point is that reducing the task to a lower dimensional problem (labeling for the supervised classification) required additional interaction, and that this feature space compression resulted in a loss of useful information (as the validation loss was better when learning from the full feature space as opposed to the compressed one).

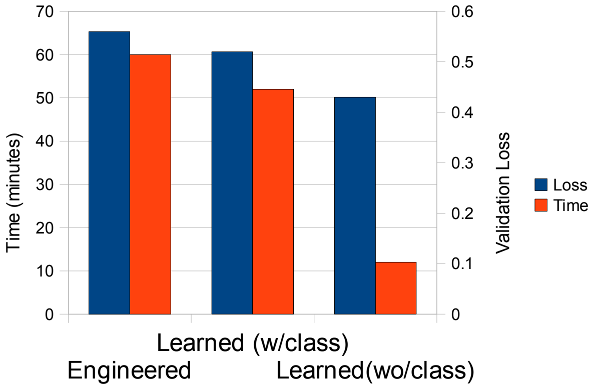


Figure 5.14: Performance comparison between 3 approaches to generating prior costmaps. LEARCH not only performs better and faster at the task of determining costs of different semantic classes, it also does a better job at interpreting raw data (for the purpose of costing) than semantic classification.

5.5.2 Learning to Interpret Online Perceptual Data

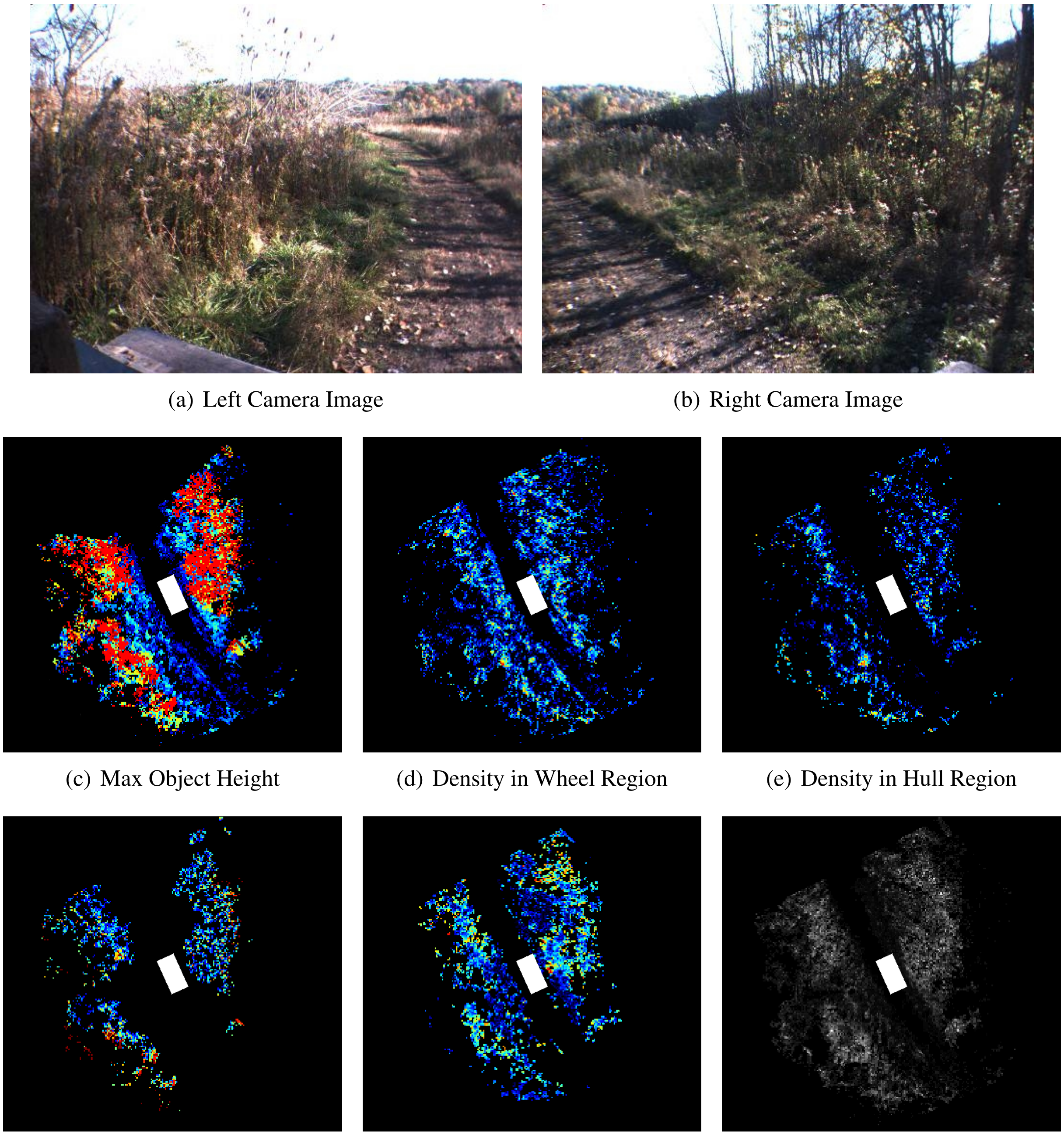
Next, DR-LEARCH was applied to the task of learning a cost function for Crusher's onboard perception system (Figure 5.15). Training data in the form of expert example behaviors was gathered by having Crusher's safety operator RC the vehicle through sets of specified waypoints. Different training examples were collected over a period of months in varying locations and weather conditions, and with 3 different operators at one time or another. During data collection, all raw sensor data was logged along with the example path. Perceptual features were then produced offline by feeding the raw sensor data through Crusher's perception software. In this way, the base perception system and its features could be modified and improved without having to recollect new training data; the raw data is just reprocessed, and a cost function learned for the new features.

This set of examples was first used to perform a series of offline experiments to validate the dynamic extension of LEARCH. Loss on a validation set was used as the metric, to measure how well planned behavior recreated example behavior. These results are presented in Figure 5.16. The left graph again demonstrates the effectiveness of performing balanced as opposed to unbalanced regression, as the balanced version has superior validation performance. The center graph further demonstrates the utility of replanning with corridor constraints. With a small corridor size, the algorithm is able to smooth out some of the noise in example human behavior, and improve generalization. As the corridor size increases, the algorithm begins to over-smooth, resulting in decreasing validation performance. This also demonstrated how validation data can be used to automatically determine the optimal corridor size.

An experiment was also performed to assess the effectiveness of filtering out inconsistent timesteps. A single expert behavior was used to learn a cost function, first with no filtering, and then with approximately 10% of its timesteps automatically filtered. As would be expected, the

[9] The vehicle's speed is controlled online based on the proximity of (perception reported) obstacles; therefore the safer the terrain the faster the vehicle will attempt to drive

[10] Neither of these ranges include the necessary effort to process raw overhead data into feature maps, as this process is a shared precursor to both approaches



(a) Left Camera Image

(b) Right Camera Image

(c) Max Object Height

(d) Density in Wheel Region

(e) Density in Hull Region

(f) Density above Vehicle

(g) Solid Classification

(h) Learned Cost

Figure 5.15: An example of learned perception costs from a simple scene, depicted in (a),(b). Processing of raw data results in perceptual features (a subset of which are shown in (c) - (g)) such as perceived object density at various heights from the ground, or classification into solid or vegetative objects. These feature are then mapped into cost (h) by a learned function.

performance on the remaining 90% of the training set improved after filtering (Figure 5.17). However, performance on the validation set also improved slightly. This demonstrates that filtering out inconsistent timesteps not only improves performance on examples for which these timesteps were

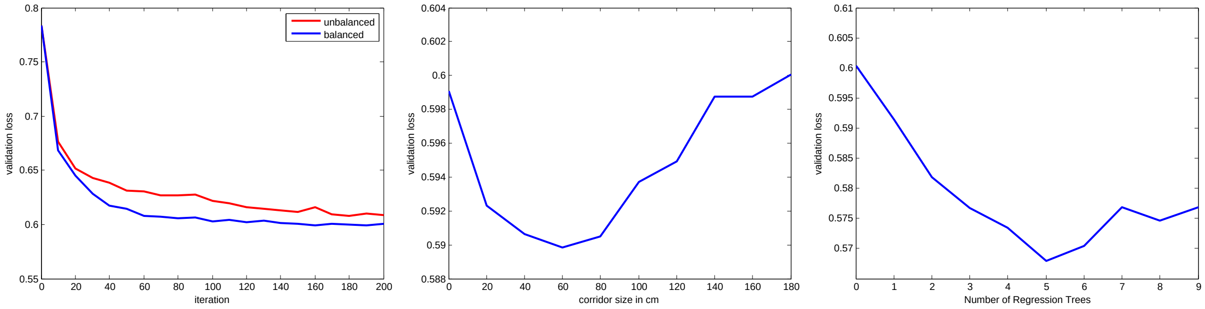


Figure 5.16: Results of offline experiments on logged perception data. **Left:** Validation Loss during learning for the balanced and standard weighting **Center:** Validation Loss as a function of the replanning corridor size **Right:** Validation Loss as a function of the number of regression trees.

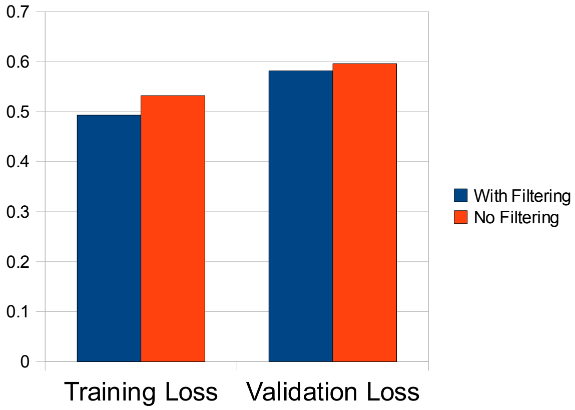
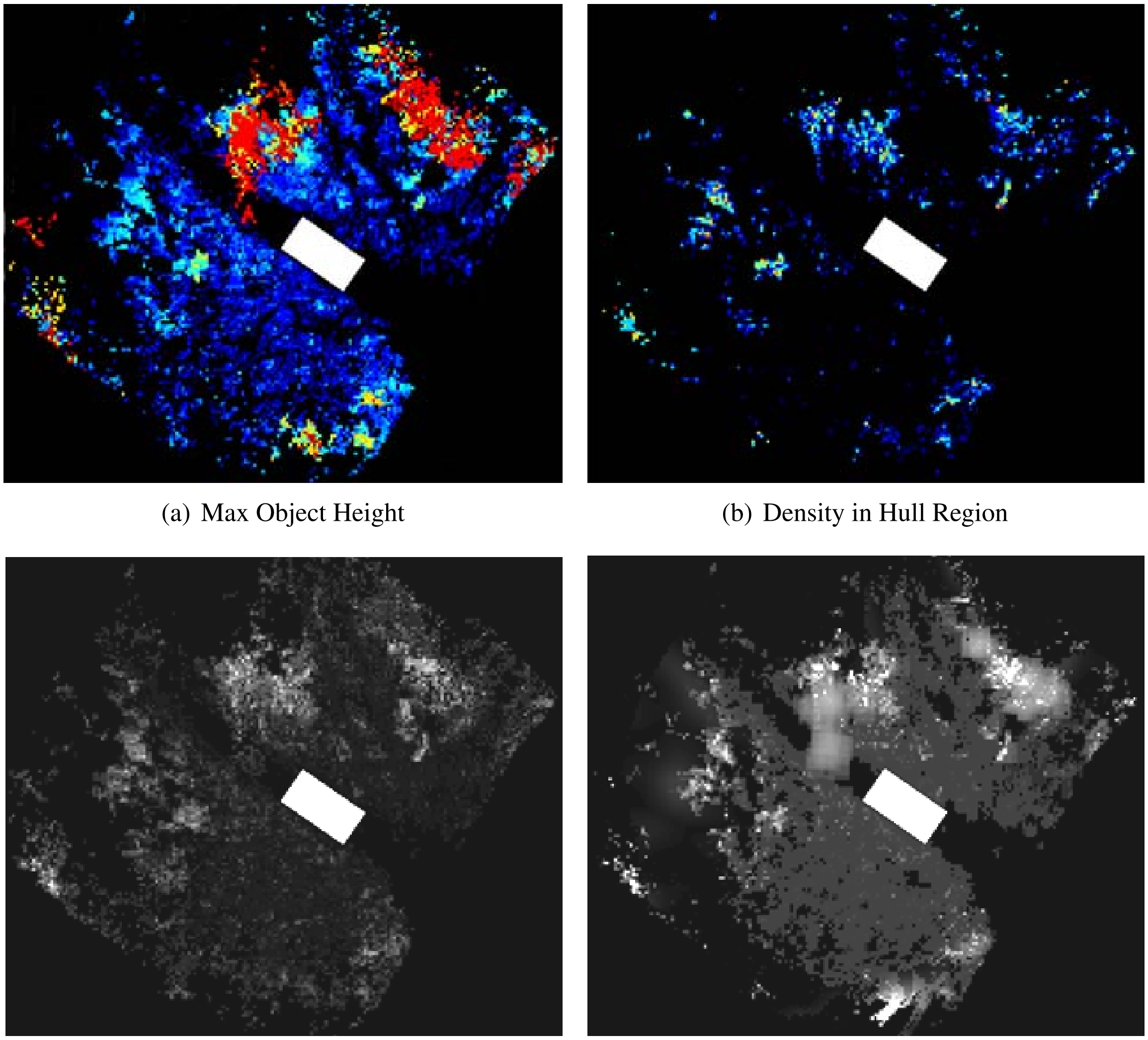


Figure 5.17: Comparison of performance with and without filtering. Filtering out inconsistent timesteps improves performance on both training example which were inconsistent with the filtered examples, as well as on an independent validation set.

known to be inconsistent, it also improves generalization to unseen examples.

As cost evaluations in an onboard perception system must be performed in real time, the computational cost of an evaluation is an important consideration. As described in (Section 5.4.1), using only linear regressors is beneficial from a computational standpoint, and feature learning can be used to improve the complexity of the cost function if necessary. Figure 5.16 (**Right**) shows validation loss as a function of the number of added features learned using simple regression trees. At first, additional features improves the validation performance; however, eventually too many features can cause overfitting.

Next, the collected training set was used to learn a cost function to run onboard Crusher. Originally, Crusher used a hand tuned perception cost function. During more than 3 years of Crusher development, this cost function was continually redesigned and retuned, culminating in a high performance system [101, 102]. However, this performance came at a high cost in human effort. Version control logs indicate that 145 changes were made to just the structure of the model mapping perceptual features to costs; additionally more than 300 parameter changes were checked in



(a) Max Object Height

(b) Density in Hull Region

(c) Learned Cost

(d) Hand Tuned Cost

Figure 5.18: A comparison of the learned vs hand tuned cost function for Crusher's perception system. In this scene, the area directly in front of Crusher contains no ground level obstacle, but does contain overhanging tree canopy. The learned cost function produces a reasonable low cost, while the hand tuned cost function produces a higher cost.

(untold more were tried at one point or another, requiring verification on logged data or through actual vehicle performance). As each committed change requires significant time to design, implement, and validate, easily hundreds of hours were spent on manually constructing the cost function. In contrast, the time to collect the final training set for learning by demonstration required only a few hours of human time (spread over several months). This seemingly small amount of human demonstration is sufficient due to the numerous constraints implied by each example behavior: the approximately 3 kilometers of demonstration provides hundreds of thousands of examples of states to traverse, and millions more examples of states that were avoided.

Crusher's perception system actually consists of two modules: a local perception system that produces high resolution cost data out to 20m, and a far range perception system that produces medium resolution cost data out to 60m. The far range system utilizes near-to-far learning as described in [103, 104] for learning a far range cost function from near range cost data. Therefore, when the system learns online from scratch, the near range cost function implicitly defines the far range cost function. For this reason, learning by demonstration was only used to learn the near range function; the far range function would automatically adapt to whatever near range function was utilized.

A comparison of the learned and hand tuned cost functions is shown in Figure 5.18. In this scene, Crusher is facing an open trail with vegetation to its left and a tree on its right. The tree canopy hangs over the trail, but does not drop down to the vehicle height. The learned cost function considers this overhang a low cost region; the associated behavior would be to continue down the trail. However, the hand tuned cost function considers this area a medium cost, enough to force Crusher temporarily off the trail. The cause of this increased cost is a hand coded rule that was intended to force Crusher to give a wider berth to large areas of high density obstacles. However, in this case the rule false positives, and incorrectly increases the cost of an easily traversable patch of terrain. This exemplifies one of the core problems with hand tuning of preference models: the inability of human experts to properly consider and validate against more than a handful of examples can lead to overfitting.

The performance of different perception cost functions was quantitatively compared through more than 150 km of comparison trials. The final results comparing 4 different cost functions are presented in Table 5.2. In addition to the hand tuned cost function, 3 learned cost functions were compared: one using the global planner, one using the local planner without boostrapping a feasible example, and one using the local planner with such bootstrapping (Section 5.4.3). Since the expert demonstrators were only concerned with generating training paths that drove over the correct terrain[11], the desired example action was estimated using the heuristic described in Section 5.4.3.

As the cost function itself is the variable being tested, cost can not be used as a metric for comparing the safety or performance of each system. Therefore, various proprioceptive statistics were recorded to offer an approximate quantitative description of safety. The number of safety related e-stops was also recorded. The statistics for each learned system were then compared on a waypoint by waypoint basis to the hand tuned system, to test for statistical significance. While it is not possible to convert these numbers into a single quantitative metric for comparison[12], it is possible to make a broad relative comparison between systems by observing individual statistics. That is, if the majority of the dimensions for comparison all provide evidence supporting the benefit of a particular system, then the relative weighting amongst such dimensions is reduced in importance.

In comparison to the hand tuned system, the cost function learned for the global planner resulted in overly aggressive behavior. As discussed in Section 5.4.3, learning in this manner does not result in sufficiently high costs for Crusher to avoid obstacles online (Figure 5.7); the empirical result is that Crusher drives faster, turns less, and backs up less, while suffering from increased mobility risk (in the form of twice the rate of safety e-stops). In contrast, the cost function learned for the local planner (without bootstrapping) resulted in performance very similar to that of the already high performance hand tuned system; The only significant difference was the learned system

[11] For example, the experts would often perform short point turns or back and forth 'wiggle' maneuvers to squeeze Crusher through tight spaces, knowing that only *where* they drove and not *how* they drove was being used for training

[12] doing so would require a manually tuned and essentially arbitrary weighting, the very problem this work seeks a solution to

System	Avg Dist	Avg Cmd	Avg Cmd	Avg Lat	Dir Switch	Avg Motor
	Made Good (m)	Vel (m/s)	Ang Vel.($\circ/s$)	Vel (m/s)	Per m	Current (A)
Hand Tuned	130.7	3.24	6.56	0.181	0.107	7.53
Global	123.8*	3.34*	4.96*	0.170*	0.081*	7.11*
Local wo/bootstrap	127.3	3.28	5.93*	0.172*	0.100	7.35
Local w/bootstrap	124.3*	3.39*	5.08*	0.170*	0.082*	7.02*

System	Avg	Avg	Avg Vert	Avg Lat	Susp	Safety
	Roll($\circ$)	Pitch($\circ$)	Accel (m/s^2)	Accel (m/s^2)	MaxΔ (m)	E-stops
Hand Tuned	4.06	2.21	0.696	0.997	0.239	0.027
Global	4.02	2.22	0.710*	0.966*	0.237	0.054*
Local wo/bootstrap	4.06	2.22	0.699	0.969*	0.237	0.034
Local w/boostrap	3.90*	2.18	0.706*	0.966*	0.234*	0.030

Table 5.2: Averages over 295 different waypoint to waypoint trials per perception system, totaling over 150km of traverse. Statistically significant differences (from hand tuned) denoted by *

turned slightly less (as indicated by lower average angular velocity and lateral speed/acceleration).

Boostrapping a feasible example path by using the cost function learned for the global planner resulted in a system that also maintained a seemingly equal level of safety to the hand tuned system; the difference in safety e-stops was not statistically significant, but there was a significant decrease in the wear on Crusher in the form of lower motor current draw and less suspension travel. However, this equal safety was achieved with seemingly more aggressiveness than the hand tuned cost function. This is indicated by a statistically significant decrease not only in angular velocity and lateral movement, but also in the amount of backing up, with a significant increase in average speed. As discussed in Section 5.4.3, the effect of the boostrapping stage is to reduce noise in the cost function; the reduction of this noise allows the vehicle to alter its behavior less in the face of false positive high cost regions, while still avoiding true obstacles. The overall result is a slight performance improvement, achieved with orders of magnitude less human effort.

Chapter 6

Learning Action Preferences from Expert Demonstration

Chapter 5 focused on the problem of deciding *where* a robot should drive; that is, on determining relative costs and preferences of various patches of terrain. However, this is not the only domain where determining such preferences for a mobile robot can be tedious and time consuming. There is also the problem of determining *how* to drive; that is, defining relative preferences over various robot actions and maneuvers. Simply determining which actions to prefer at first may appear as more an issue of style than safety or performance. In structured environments, this is true to a degree. For example, if it is well known where a robot can and cannot traverse, than safety can be further insured by limiting allowable actions to those that are inherently safe (e.g. no hard turns).

However, in semi or unstructured environments, this is no longer the case. While ideally a robot would never put itself in a situation where there is no inherently safe action available, this can happen due to uncertainty in both perception and control. For example, imagine a robot traveling at high speed that detects a dangerous rock. Ideally, a soft swerve action would be available to avoid the rock. But what if the rock is detected too late? Now the only options available are a hard swerve (that may miss the rock but could roll the vehicle), to attempt a hard break (that may involve a loss of control) or to hit the rock (that may result in vehicle damage). Making such decisions requires weighing preferences and tradeoffs not just over terrain patches, but also the various available actions. Therefore, ensuring safety requires not just proper modeling of the consequences actions, but also proper preference modeling. This issue becomes more complex when additional considerations aside from safety are added. For instance, there may be rules of the road to obey with varying degrees of importance. There may be a desire to ensure a smooth ride (e.g. if an automated vehicle has passengers, or to reduce wear on the vehicle). There may also be issues of performance, that is weighing the tradeoffs involved in driving faster or slower over certain terrain types. Figure 6.1 demonstrates how differing preferences for or against certain actions can result in very different end robot behavior, even in a very simple scenario.

As with terrain costing, the issue of learning preferences over planner actions for mobile robots has not received much focus. When it has, it has mostly been in the context of control and path tracking [176, 217, 218, 209, 241, 242]. A notable exception is [243], which investigated the use of learning through demonstration for deriving different driving styles for an Urban Challenge vehicle; however, this work took place in simulation. Additional recent work [244, 245] has explored the idea of learning behavior styles for a robot or autonomous agent that take into account social

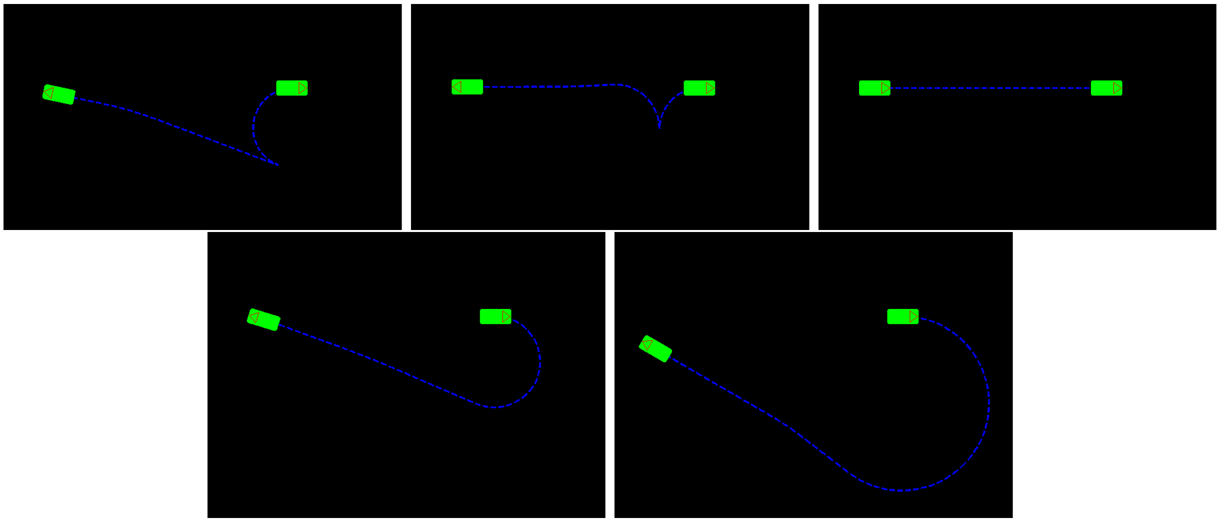


Figure 6.1: Examples of different ways a robot could drive to a goal directly behind it, each one implying a different preference for forward vs reverse, hard turns vs soft turns, etc.

considerations [129]; this work has also been confined to simulation to date. However, while the planner specific problem has received at least some focus, the wider problem of building consistent preference models over both terrains *and* actions has not been previously addressed. As in the case of fusing multiple sensors, various preference models need to be constructed in parallel, in order to ensure correct behavior over a range of scenarios.

6.1 Extending LEARCH to State-Action Pairs

The derivation of the MMP framework in Section 5.1 considered cost functions that were only defined over states, and defined a plan as a sequence of states. However LEARCH can be extended to costs over states and actions simply by treating each state-action pair as a planning state. Instead of generating plans through $\mathcal{S}$, plans are now generated through $\mathcal{S} \times \mathcal{A}$ for some space of actions $\mathcal{A}$. A path P is now a sequence of tuples $(s, a),\ s \in \mathcal{S},\ a \in \mathcal{A}$. This new definition of a path also requires a new definition of a loss function, which must now compare elements of $\mathcal{S} \times \mathcal{A}$.

Additionally two features spaces must now be considered : one defined over states $\mathcal{F}_\mathcal{S}$, and one defined over actions $\mathcal{F}_\mathcal{A}$. Therefore, for every $(s, a) \in P$, consider the associated feature vectors $F_s \in \mathcal{F}_\mathcal{S}$ and $F_a \in \mathcal{F}_\mathcal{A}$, and define the cost as $C(F_s, F_a)$ where $C : \mathcal{F}_\mathcal{S} \times \mathcal{F}_\mathcal{A} \to \mathbb{R}^+$.

By simple substitution (5.1.10) becomes

$$\text{minimize}\ \ \mathcal{O}[C] = \lambda\textsc{Reg}(C) + \sum_{(s,a)\in P_e} C(F_s, F_a) - \min_{\hat{P}} \left[\sum_{(s,a)\in \hat{P}} (C(F_s, F_a) - L_e(s, a)) \right] \tag{6.1.1}$$

A new derivation of the various LEARCH algorithms could follow straightforwardly from this objective functional. However, the dimensionality of the feature space would be an issue. Specifically, it would be hard for a cost function to generalize well over $\mathcal{F}_\mathcal{S} \times \mathcal{F}_\mathcal{A}$. For example, if an expert behavior involves a particular (s_e, a_e), instead of the hypothesis (s_*, a_*) it would be vary hard for LEARCH to determine if this decision was made with respect to the states or the actions. In order to make a well-informed decision, LEARCH would require additional demonstrations over similar terrain with different action choices, and vice versa. While large degrees of expert demonstration may provide numerous examples throughout this higher dimension space, the number of meaningful constraints would also have to be considerably higher. Therefore learning cost functions defined over $\mathcal{F}_\mathcal{S} \times \mathcal{F}_\mathcal{A}$, while feasible, would require a very large amount of carefully chosen training data that could be very time consuming to collect.

Strictly enforcing linearity in cost functions would be one possible solution to this problem, as it would remove the possibility of interdependencies between state and action features. However, this would negate the non-linear advantage of LEARCH. Instead, one could consider enforcing that there be no non-linear dependencies between state and action features. Therefore, a partitioning of the feature space is proposed. Rather than consider general cost functions over the feature space $\mathcal{F}_\mathcal{S} \times \mathcal{F}_\mathcal{A}$, only cost functions of the form $C(F_s, F_a) = C_s(F_s) + C_a(F_a)$ will be considered. That is, a separate cost function can be learned over states (terrains) and actions (maneuvers). This partitioning will allow better generalization over state and action features independently, and in turn will require less training data.

Along with concerns about training time and data collection, there is an even more practical reason for enforcing such a partitioning: it adheres to the architecture that most mobile robotic systems currently utilize. Features describing patches of terrain are usually generated by one or more perception components, while features describing various actions are generated by one or more planning components. Planning components then need cost information over both states and actions. If cost was generated by one cost function with non-linear interdependencies, planning components would need all raw perceptual feature data; given the complexity and dimensionality of perceptual features that are often generated, this could easily result in two orders of magnitude more information sharing between components [1]. Compressing all relevant information into a single cost value per state solves this problem.

However, there may at times be individual perceptual features that are useful for determining preferences over actions. Features relevant to speed control are an obvious case; certain terrain types (such as ditches) may be easily traversable at low speeds but quite dangerous at high speeds. A single cost value cannot encode this distinction. However, there is no reason that certain perceptual features could not be considered as part of both feature vectors[2]. Therefore, $\mathcal{F}_\mathcal{S}$ and $\mathcal{F}_\mathcal{A}$ should not be considered as strictly independent feature spaces, but rather simply as the features provided to each preference model.

With this partitioning of the feature space, the new objective is

$$\text{minimize } \mathcal{O}[C_s, C_a] = \lambda_s \text{REG}(C_s) + \lambda_a \text{REG}(C_a) + \sum_{(s,a)\in P_e} C_s(F_s) + C_a(F_a) \\ - \min_{\hat{P}} \left[\sum_{(s,a)\in\hat{P}} (C(F_s) + C_a(F_a) - L_e(s) - L_e(a)) \right] \qquad (6.1.2)$$

[1] Such information sharing was at one point desired for Crusher in order to allow certain forms of online learning; however, the necessary data rate was calculated to almost completely saturate the available communication bandwidth

[2] Features derived from a physical simulation would be obvious candidates

As the C_s and C_a terms are independent, this leads to independent functional gradients[3]

$$\begin{aligned}\nabla \mathcal{O}_F[C_s] &= \sum_{(s,a)\in P_e} \delta_F(F_s) \quad - \quad \sum_{(s,a)\in P_*} \delta_F(F_s) \\ \nabla \mathcal{O}_F[C_a] &= \sum_{(s,a)\in P_e} \delta_F(F_a) \quad - \quad \sum_{(s,a)\in P_*} \delta_F(F_a) \qquad (6.1.3)\end{aligned}$$

$$P_* = \arg\min_P \sum_{(s,a)\in P} (C_s(F_s) + C_a(F_a) - L_e(s) - L_e(a))$$

Each independent functional gradient can be projected onto its own direction set as in (5.1.13), leading to two new regressors R_s and R_a defined over $\mathcal{F}_\mathcal{S}$ and $\mathcal{F}_\mathcal{A}$ respectively. From this point onward, straightforward application of the LEARCH algorithm with functional gradient descent would yield cost functions of the form

$$\begin{aligned}C(F_s, F_a) &= \sum_i \eta_i [R_s^i(F_s) + R_a(F_a)] \\ &= \sum_i \eta_i R_s^i(F_s) \quad + \quad \sum_i \eta_i R_a^i(F_a) \\ &= C_s(F_s) + C_a(F_a)\end{aligned}$$

However, if instead of performing straight gradient descent, exponentiated functional gradient descent were considered, this would yield cost functions of the form

$$\begin{aligned}C(F_s, F_a) &= \exp(\sum_i \eta_i [R_s^i(F_s) + R_a(F_a)]) \\ &= \exp(\sum_i \eta_i R_s^i(F_s) \quad + \quad \sum_i \eta_i R_a^i(F_a)) \\ &= \exp(\sum_i \eta_i R_s^i(F_s)) \exp(\sum_i \eta_i R_a^i(F_a)) \\ &= C_s(F_s) C_a(F_a) \qquad (6.1.4)\end{aligned}$$

Applying exponentiated functional gradient descent results in a final cost function that is the result of multiplying the state and action terms, instead of simply adding them. Multiplying cost functions has certain advantages over adding them, as it preserves relative differences, instead of absolute differences. Multiplying independent state and action cost functions would insure that the cost ratio of action a_1 to a_2 is independent[4] of the terrain features encountered, and vice versa. This is important for generalization; that is if (all else being equal) a_1 is preferable to a_2, maintaining the relative costs ensures this preference will also be reflected (when the actions traverse similar terrain).

The above formulation offers the possibility of learning cost functions both over states and actions via a single set of expert demonstrations. That is, expert demonstration could be used to learn how to both interpret perceptual data, while at the same time demonstrating the proper

[3] as before regularization is ignored for clarity

[4] As before, specific terrain dependencies can be accounted for by augmenting the action feature vector

tradeoffs with respect to various actions. Such an approach would require little to no additional expert demonstration beyond that which is already necessary for learning to interpret perceptual data.

6.2 Learning Planner Preference Models

Without loss of generality, first consider the case where C_s is known a priori. Given a known C_s, equation (6.1.3) reduces to

$$\nabla \mathcal{O}_F[C_a] = \sum_{a \in P_e} \delta_F(F_a) \quad - \sum_{(s,a) \in P_*} \delta_F(F_a) \tag{6.2.1}$$

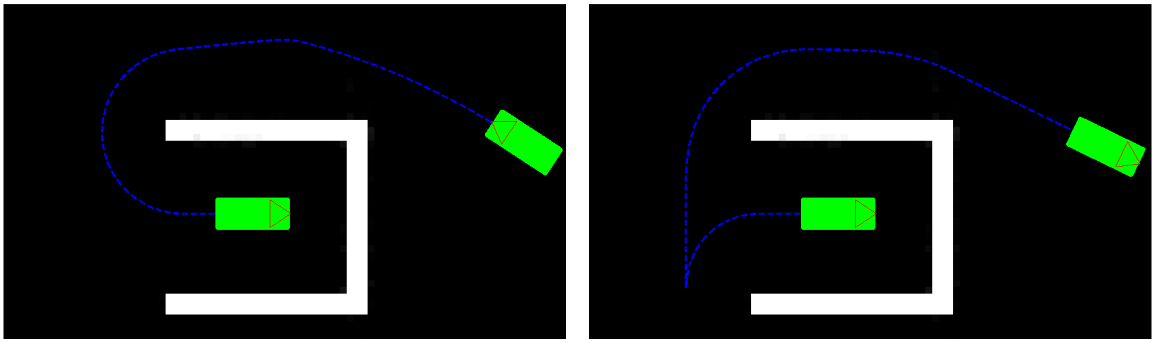
$$P_* = \arg\min_P \sum_{(s,a) \in P} \left(C_s(F_s) + C_a(F_a) - L_e(s) - L_e(a)\right)$$

As before, the functional gradient indicates that to minimize the objective, the cost of actions in the example plan should be lowered, and the cost of actions in the current plan should be raised. In the same manner as equation (5.1.13), R_a can be computed as a projection of the functional gradient, and used as an update step for exponentiated functional gradient descent.

This formulation assumes that a plan consists of a sequence of state-action tuples in $\mathcal{S} \times \mathcal{A}$ that lead all the way from a start state to a goal state. Previous formulation have general assumed that $\mathcal{S} = \mathbb{R}^n$; that is, that the state of the robot was simply its location. However, a motion planner operating purely in this state space will not reliably generate feasible plans for a mobile robot (assuming it has non-holonomic constraints). At the very least, motion planners for ground vehicles will often operate in SO(2) to take into consideration the vehicle's current heading. Augmenting state with the current linear and angular velocity is also common; augmentation with accelerations is becoming more so. Efficient motion planning in such higher dimensional kinodynamic state spaces is more complex and expensive, and is an entire field of research unto itself [36, 37, 155, 246].

6.2.1 Receding Horizon Motion Planners

The added cost of planning in high dimensions can cause problems when motion planning must be performed in real time at a high rate. The most common solution is to only compute motion plans out to a certain distance or time horizon. A heuristic value is used from the end of each possible plan to estimate the remaining cost-to-go. This approach is similar to the idea of receding horizon control, and is based on the same logic; namely, it is extremely unlikely that a complex trajectory will ever be executed exactly as planned (due to control error and/or state estimation error); therefore future sections of a trajectory will need to be replanned anyway. In addition, in the mobile robot domain direct sensing range is generally limited to some horizon; therefore any plan outside said horizon is using stale information. Such an approach has been used implicitly for quite some time. For example, the behavior based approach of the ALV [42, 43] would execute a single action aimed at minimizing a heuristic cost-to-go (in this case distance). Over time, this



(a) No Preference Model (following heuristic)

(b) Demonstrated Desired Behavior

Figure 6.2: An example where just following the heuristic of a single arc motion planner leads to undesirable behavior. Repeatedly choosing reverse arcs minimizes the cost-to-go at each timestep, as the cost-to-go does not reflect the cost of driving in reverse. Especially for robots that only have sensors in front, driving forward is heavily preferred.

approach has become more formalized. In addition, more complex heuristics have been used including using the output of a lower dimensional planner [95].

Crusher's planning system described in Section 5.4.3 follows exactly this setup. The local motion planner evaluates a finite set of a actions, in this case constant curvature arcs, out to a set horizon. Field D* is used as a heuristic to estimate the cost-to-go from the state at the end of each action. As previously described such single arc motion planners, while simplistic, have proven quite effective for mobile robots in complex terrain [95, 101, 102, 124, 120, 121, 122, 127]. However, the robust operation of such a planning system is often dependent on a well tuned model specifying preference over possible planner actions. For example, it is common to prefer softer turns to harder ones (all else being equal), or to prefer driving forward than in reverse (especially if the robot does not have rear sensors).

Aside from issues of driving style (which generally affect robot performance at the margin), planner preference models can also affect the most basic behavior of a robot. For example, imagine the simple scenario show in Figure 6.2. In such a case, the lowest cost action according to the heuristic is to drive in reverse; this action will be repeated all the way to the goal. Even if the planner preference model heavily penalizes reverse actions in favor of forward ones, the reverse actions could still have a cost advantage. The source of this issue is the cost-to-go from the end of each arc. If the cost-to-go were accurate, than it would reflect either the cost of turning around to drive forward to the goal, or of driving in reverse the entire way. However, the cost-to-go is just a heuristic (under this architecture the Field D* cost), and does not take this additional turning into account. Computing the exact cost-to-go would require solving the original high dimensional planning problem, the difficulty of which was the reason for a receding horizon approach in the first place. The end result is that simply choosing actions that minimize the cost-to-go will not always achieve desirable behavior

An alternative solution to solving the highest dimensional planning problem is to use a planner for the heuristic that operates in at least SO(2), but not necessarily the full state space. In practice, there is nothing wrong with this approach as long as it is computationally feasible, and it has been demonstrated on fielded systems [36, 247]. However, without solving the full dimensional

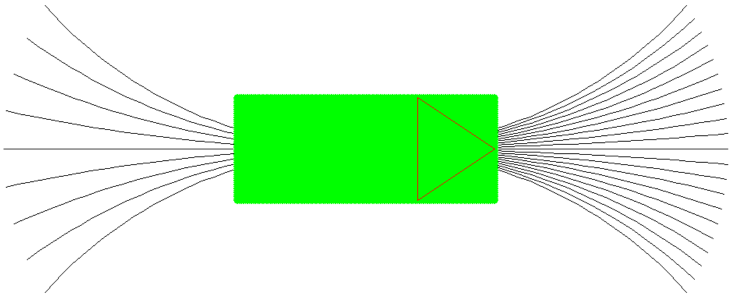


Figure 6.3: An action set for a single arc motion planner, with 21 forward arcs and 9 reverse arcs.

planning task, there will always remain a possibility of undesirable behavior due to this mismatch. Therefore, the rest of this chapter will use Crusher's single arc motion planner with a Field D* heuristic as a proof of concept for higher dimensional receding horizon planning architectures. The high performance achieved with such local/global hybrid architectures on multiple robotic vehicles validates this challenge as sufficient from which to infer meaningful results; that is, preference model construction techniques that work on this planning architecture will generalize to more complex ones.

6.2.2 Learning Preference Models for the Single Arc Motion Planner

The single arc motion planner can be formalized as follows. Each arc is parameterized by its direction[5] (forward or reverse) and its curvature, resulting in an action space of

$$\mathcal{A} = \{-1,\ 1\} \times [-\text{MAXCURV}, \text{MAXCURV}]$$

where MAXCURV is the maximum curvature of the vehicle. In practice, rather than a continuous parameterization over curvature, a discrete set of arcs is chosen in this interval. Each action consists of following the associated arc to a set horizon. Figure 6.3 shows an example action set.

Given a current state x and a goal state g, the local planner chooses the optimal action $A_* \in \mathcal{A}$ according to

$$A_* = \arg\min_{a \in \mathcal{A}} \ C_a(F_a)[\sum_{s \in T^{a,x,g}} C_s(F_s)] \tag{6.2.2}$$
$$T^{a,x,g} = \{s \in \text{ARC}(a,x)\} \ \bigcup \ \{s \in \text{DSTAR}(\text{ARCEND}(a,x), g)\}$$

$T^{a,x,g}$ is the set of states along the trajectory that the planner expects to follow should it choose action a. It consists of the the set of states along arc a, along with the states along the optimal D* path from the end of a to the goal. The sum state based cost is multiplied by the action of the (single) action $C_a(F_a)$ as in equation (6.1.4).

If a desired example action A_e is provided at x, and C_s is assumed to already be known, then

[5] For single arc motion planners, velocity control is often decoupled from arc selection

the standard MMP (hard) constraint can be written as

$$\left((C_a(F_{A_*}) - L_e(A_*)) + \sum_{s \in T^{A_*,x,g}} C_s(F_s) \right) \geq \left(C_a(F_{A_e}) + \sum_{s \in T^{A_e,x,g}} C_s(F_s) \right) \tag{6.2.3}$$

As in Section 5.1, adding a margin term and accounting for the possibility of not being able to meet all constraints via slack penalties results in a constrained optimization similar to (5.1.5)

$$\text{minimize } \mathcal{O}[C_a] = \lambda \textsc{Reg}(C_a) + \zeta \tag{6.2.4}$$

subject to the constraint

$$\left((C_a(F_{A_*}) - L_e(A_*)) + \sum_{s \in T^{A_*,x,g}} C_s(F_s) \right) - \left(C_a(F_{A_e}) + \sum_{s \in T^{A_e,x,g}} C_s(F_s) \right) + \zeta \geq 0$$

moving the constraints into the optimization yields an objective functional mirroring (5.1.10) and (6.1.1)

$$\mathcal{O}[C_a] = \lambda \textsc{Reg}(C_a) + \left(C_a(F_{A_e}) + \sum_{s \in T^{A_e,x,g}} C_s(F_s) \right) - \left((C_a(F_{A_*}) - L_e(A_*)) + \sum_{s \in T^{A_*,x,g}} C_s(F_s) \right) \tag{6.2.5}$$

Although state and action costs are multiplied in the planner's internal optimization, they are added in this objective functional; as in (6.1.2) - (6.1.4) this is consistent due to the use of exponentiated functional gradient descent.

In this context, L_e is defined over $\mathcal{A}$. Given the more complex definition of $\mathcal{A}$ the construction of L_e is not quite as obvious as in the previous chapter, when L_e was based on the L_2 norm. Instead, in this case L_e is defined as

$$L_e(A) = L(A_e, A) = |A - A_e|_\infty \tag{6.2.6}$$

That is, the loss value is the max of the error in direction and the error in curvature. Depending on the value of MAXCURV (and the associated range of the curvature dimension), an additional weighting could be used to normalize the two dimensions.

This objective function leads to a very straightforward gradient

$$\nabla \mathcal{O}_F[C_a] = \delta_F(F_{A_e}) \;-\; \delta_F(F_{A_*}) \tag{6.2.7}$$

Simply stated, to minimize the objective, the cost of the example action should be lowered, and the cost of the current optimal action should be raised. This (sub)gradient descent should be repeated until the constraint in (6.2.3) is satisfied.

These equations naturally give rise to a learning from demonstration algorithm for automatic construction of planner preference models. This algorithm is very similar to D-LEARCH in Section 5.2, and is similarly based off of discretizing an expert's actual demonstration into a set of static examples. Formally, this results in the return of the time index t to account for dynamics. As opposed to a start state x, x^t now represents the state at time t. The new objective and its functional

Algorithm 6: D-LEARCH for planner preference models with known perception cost

```
Inputs : Example Behaviors $P_e^1, P_e^2, ..., P_e^n$, Sensor Histories $\mathcal{H}^1, \mathcal{H}^2, ..., \mathcal{H}^n$, Perception
         Cost Function $C_s$
$C_{a_0} = 1$;
foreach $P_e^i$ do
    for $\tau$ =firstTime($P_e^i$):$\Delta\tau$:lastTime($P_e^i$) do
        $P_e^{\tau,i}$ = extractPathSegment($P_e^i$,$\tau$,lastTime($P_e^i$));
        $A_e^{\tau,i}$ = extractFirstAction($P_e^i$,$\tau$);
        [$\mathcal{F}_s^{\tau,i}$, $\mathcal{V}^{\tau,i}$] = simulatePerception($\mathcal{H}^i$,firstTime($P_e^i$),$\tau$);
for $j = 1...K$ do
    $T_f = T_o = \emptyset$;
    foreach $P_e^{t,i}$ do
        $\mathcal{M}^{t,i}$ = buildCostmap($C_s$,$\mathcal{F}_s^{t,i}$,$\mathcal{V}^{t,i}$);
        $A_*^{t,i}$ = chooseLossAugAction(start($P_e^{t,i}$),goal($P_e^{t,i}$),$\mathcal{M}^{t,i}$,$C_{a_{j-1}}$);
        if $A_*^{t,i} \neq A_e^{t,i}$ then
            $\mathcal{F}_a^{t,i,e}$ = computeActionFeatures($A_e^{t,i}$,$\mathcal{M}^{t,i}$);
            $\mathcal{F}_a^{t,i,*}$ = computeActionFeatures($A_*^{t,i}$,$\mathcal{M}^{t,i}$);
            $T_f = T_f \bigcup \mathcal{F}_a^{t,i,e} \bigcup \mathcal{F}_a^{t,i,*}$;
            $T_o = T_o \bigcup -1 \bigcup 1$;
    $R_j$ = trainRegressor($T_f$,$T_o$);
    $C_{a_j} = C_{a_{j-1}} * e^{\eta_j R_j}$;
return $C_{a_K}$
```

gradient are

$$\mathcal{O}[C_a] = \lambda \text{REG}(C_a) \; + \sum_t \left(C_a(F^t_{A^t_e}) + \sum_{s \in T^{A^t_e, x^t, g, t}} C_s(F^t_s) \right) \tag{6.2.8}$$

$$- \sum_t \left((C_a(F^t_{A^t_*}) - L^t_e(A^t_*)) + \sum_{s \in T^{A^t_*, x^t, g, t}} C_s(F^t_s) \right)$$

$$\nabla \mathcal{O}_F[C_a] = \sum_t \left(\delta_F(F^t_{A^t_e}) \quad - \quad \delta_F(F^t_{A^t_*}) \right)$$

As in Section 5.2 the gradient contributions of multiple timesteps (and multiple demonstrations) are simply combined into a single gradient boosting step, and a single weighted regressor added to the previous cost function. This procedure is shown in detail in Algorithm 6.

6.2.3 Correcting for the Receding Horizon: Slack Re-scaling

The optimization presented in (6.2.8) and its implementation in Algorithm 6 will result in a cost function that considers a specified example action optimal, provided such a cost function exists. However, just as with learning perception cost functions, problems can arise when multiple example actions are provided. Since a single demonstrated trajectory is the concatenation of multiple example actions over time, any real implementation of this approach will involve trying to match a large set of example actions.

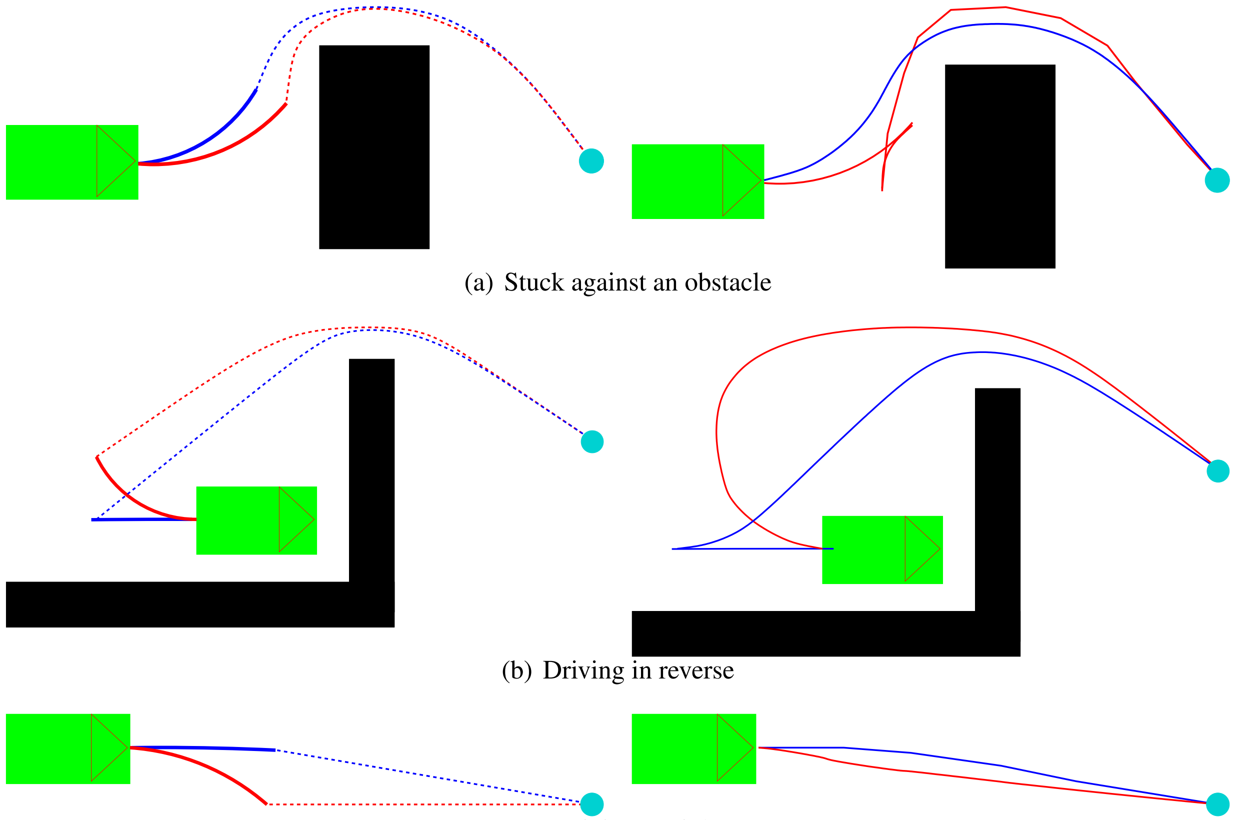
When multiple example actions are considered at once, the overall optimization will seek to minimize the sum (or equivalently the average) of their associated cost differences (between the example action and its corresponding planned action). In addition, the loss bound implied by this optimization (Theorem 5.1.1) is also on the sum of loss over individual actions. Unfortunately, this leaves open the possibility of poor performance for a subset of examples. Especially when learning from noisy and imperfect expert demonstration, it is always possible that certain individual examples will not be properly learned.

Unfortunately, certain errors can be more costly than others. In the case of perception, such costly errors would be reflected in the resulting plan; for example, if a dangerous obstacle was not receiving sufficient cost, than the current planned behavior would continue to pass through it, implying that the cost still needed to be increased. However, this is not the case with the single arc planner. Figure 6.4 shows examples of such scenarios. In Figure 6.4(a), the example action is a slightly harder turn than the current planned action, resulting in nonzero but small loss. However, by executing the wider turn, the robot will become stuck against the obstacle, and will be forced to back up and turn to get around it. Figure 6.4(b) presents an even more extreme case. By backing up from an obstacle, but turning the wrong way, the robot ends up executing a very different trajectory; the path is similar, but it is executed in reverse. The reverse scenario can also occur. Figure 6.4(c) shows a case where there is nonzero loss between the demonstrated and planned action, but the resulting trajectories are nearly identical. However, even though this error is of almost no consequence, the optimization will still expend effort to correct it, possibly impeding the progress of other, more important examples. This issue is not just theoretical; it can result in quite undesirable actual robot performance (see Figure 6.8).

The major contributor to this issue is the quite limited horizon of the single arc planner, which prevents it from having useful information about the consequences of certain actions. For example, in Figure 6.4(a) the obstacle is just outside the planner horizon (i.e. the arc length) and is therefore not directly aware of the trouble it could get into. The heuristic also underestimates the severity of the problem (since Field D* is not kinematically constrained, the cost is only slightly higher). While the single arc planner represents the extreme of this problem, similar horizon effects can occur with any receding horizon planner. Therefore, what is needed is a formalism to indicate that certain examples are more important than others (in that getting them wrong will have more significant consequences) and treat them as such in the optimization. Such a formalism could also minimize the importance of examples where a wide range of actions result in near identical future behavior, indicating that it is not necessary to get the example action exactly correct.

Tsochantaridis et al. present such a formalism in [248] in the context of support vector machines. One proposed solution is to scale the margin size in proportion to the loss incurred; however, this adaptation is already a part of the MMP framework (Equation 5.1.4) and is responsible for the loss bound (Theorem 5.1.1). Another proposed solution is the idea of slack re-scaling, in which the slack penalties are weighted in the optimization relative to loss or some other desired penalty function. If this penalty function were chosen properly, it could offer a solution to these negative horizon effects.

Let $\mathcal{P}_e(A)$ represent a generic error or penalty function between A_e and A under C_a. Aside from strict positivity, the only other requirement on $\mathcal{P}_e(A)$ is that $\mathcal{P}_e(A) = 0 \iff L_e(A) = 0 \iff$



(a) Stuck against an obstacle

(b) Driving in reverse

(c) Driving straight

Figure 6.4: Examples of scenarios where error between actions (left) and final behavior (right) do not correspond. Demonstrated behavior is in blue, and planner behavior is in red.

$A = A_e$.

Slack rescaling would modify (6.2.4) as such

$$\text{minimize } \ \mathcal{O}[C_a] = \lambda\text{R}\textsc{eg}(C_a) + \mathcal{P}_e(A_*^{\mathcal{P}})\zeta \tag{6.2.9}$$

subject to the constraint

$$\left((C_a(F_{A_*^{\mathcal{P}}}) - L_e(A_*^{\mathcal{P}})) + \sum_{s\in T^{A_*^{\mathcal{P}},x,g}} C_s(F_s)\right) - \left(C_a(F_{A_e}) + \sum_{s\in T^{A_e,x,g}} C_s(F_s)\right) + \zeta \geq 0$$

$$A_*^{\mathcal{P}} = \arg\min_{A} \mathcal{P}_e(A)\left((C_a(F_A) - L_e(A)) + \sum_{s\in T^{A,x,g}} C_s(F_s)\right)$$

Aside from the additional weighting of the slack penalties, the other caveat is the replacement of A_* (the action that minimizes the current cost) with $A_*^{\mathcal{P}}$ (the action that minimizes the current penalty weighted cost).

Alternatively, the slack can be scaled by the inverse penalty in the constraint

$$\text{minimize } \mathcal{O}[C_a] = \lambda\textsc{Reg}(C_a) + \zeta \tag{6.2.10}$$

subject to the constraint

$$\left((C_a(F_{A_*^{\mathcal{P}}}) - L_e(A_*^{\mathcal{P}})) + \sum_{s \in T^{A_*^{\mathcal{P}},x,g}} C_s(F_s)\right) - \left(C_a(F_{A_e}) + \sum_{s \in T^{A_e,x,g}} C_s(F_s)\right) + \frac{\zeta}{\mathcal{P}_e(A_*^{\mathcal{P}})} \geq 0$$

From this second formulation, a relationship between $\mathcal{P}_e$ and L_e can be derived.

Theorem 6.2.1. *For Loss $L_e(A_*^{\mathcal{P}})$ and Penalty $\mathcal{P}_e(A_*^{\mathcal{P}})$ the bound*

$$L_e(A_*^{\mathcal{P}})\mathcal{P}_e(A_*^{\mathcal{P}}) \leq \zeta \tag{6.2.11}$$

will always hold

Proof. Let $\mathcal{C}_e = \left(C_a(F_{A_e}) + \sum_{s \in T^{A_e,x,g}} C_s(F_s)\right)$, $\mathcal{C}_* = \left(C_a(F_{A_*^{\mathcal{P}}}) + \sum_{s \in T^{A_*^{\mathcal{P}},x,g}} C_s(F_s)\right)$

Consider the following cases

Case 1 $A_*^{\mathcal{P}} = A_e$

Then trivially $\zeta = L_e(A_*^{\mathcal{P}}) = \mathcal{P}_e(A_*^{\mathcal{P}}) = 0$

Case 2 $A_*^{\mathcal{P}} \neq A_e$

The constraint in (6.2.10) can be rewritten as

$$\mathcal{C}_* - \mathcal{C}_e \geq L_e(A_*^{\mathcal{P}}) - \frac{\zeta}{\mathcal{P}_e(A_*^{\mathcal{P}})} \tag{6.2.12}$$

By definition, $\mathcal{C}_e \geq \mathcal{C}_*$, which implies

$$\begin{aligned} \mathcal{C}_e \geq \mathcal{C}_* &\implies \mathcal{C}_* - \mathcal{C}_e \leq 0 \\ &\implies L_e(A_*^{\mathcal{P}}) \leq \frac{\zeta}{\mathcal{P}_e(A_*^{\mathcal{P}})} \\ &\implies L_e(A_*^{\mathcal{P}})\mathcal{P}_e(A_*^{\mathcal{P}}) \leq \zeta \end{aligned}$$

□

Equation (6.2.11) demonstrates how adding slack re-scaling bounds the additional penalty function. Essentially, as opposed to simply bounding the loss (as in Theorem 5.1.1), two different error functions (the original loss and the new penalty) can be partially bounded (as long as $\mathcal{P}_e = 0 \iff L_e = 0$). The caveat is that there is a tradeoff between the two error functions; that is, one error will be allowed to be high as long as the other error function is low.

However, this new formulation is concerned with $A_*^{\mathcal{P}}$, as oppossed to the original A_*. That is, it requires a planner that can solve the new argmin by being aware of the penalty function $\mathcal{P}_e$. This is undesirable for two reasons. The first is that it does not achieve the stated goal of learning a cost function for the planner that will actually run on a robot, which only minimizes cost and not the penalty weighted version [6]. The second reason is computational. A penalty aware planner must compute the penalty for every possible action. If this operation is expensive (as it will prove to be for the penalty function used in this work) then a penalty aware planner would prove prohibitively slow.

Fortunately, the benefits of slack rescaling can still be achieved using the original planner. This can be shown by looking at the optimizations that would actually be run using both planners. First, consider the objective if the tight constraints from (6.2.9) are moved into the objective

$$\begin{aligned}\mathcal{O}^{\mathcal{P}}[C_a] = \lambda\textsc{Reg}(C_a) \ &+ \mathcal{P}_e(A_*^{\mathcal{P}}, C_a)\left(C_a(F_{A_e}) + \sum_{s\in T^{A_e,x,g,t}} C_s(F_s)\right) \\ &- \mathcal{P}_e(A_*^{\mathcal{P}}, C_a)\left((C_a(F_{A_*^{\mathcal{P}}}) - L_e(A_*^{\mathcal{P}})) + \sum_{s\in T^{A_*^{\mathcal{P}},x,g,t}} C_s(F_s)\right)\end{aligned} \tag{6.2.13}$$

Next, consider the same basic objective function, but with A_* instead.

$$\begin{aligned}\mathcal{O}[C_a] = \lambda\textsc{Reg}(C_a) \ &+ \mathcal{P}_e(A_*, C_a)\left(C_a(F_{A_e}) + \sum_{s\in T^{A_e,x,g,t}} C_s(F_s)\right) \\ &- \mathcal{P}_e(A_*, C_a)\left((C_a(F_{A_*}) - L_e(A_*)) + \sum_{s\in T^{A_*,x,g,t}} C_s(F_s)\right)\end{aligned} \tag{6.2.14}$$

That is, the terms are still penalty weighted, but the original planner is used.

Theorem 6.2.2. *The objective functions defined in* (6.2.13) *and* (6.2.14) *share a solution.*

Proof. The minimum of $\mathcal{O}[C_a]$ occurs when $A_* = A_e$ and C_a is maximally regularized. Since $A_* = A_e$ imples that $L_e(A_*) = \mathcal{P}_e(A_*) = 0$, any solution that minimizes $\mathcal{O}[C_a]$ will also minimize $\mathcal{O}^{\mathcal{P}}[C_a]$ □

Theorem 6.2.3. *The gradients of the objective functions defined in* (6.2.13) *and* (6.2.14) *are always correlated.*

Proof. The gradient of (6.2.14) is

$$\nabla\mathcal{O}_F[C_a] = \mathcal{P}_e(A_*)(\delta_F(F_{A_e}) \ - \ \delta_F(F_{A_*}))$$

This gradient equals $\mathcal{P}_e(A_*)$ at $F = F_{A_e}$, $-\mathcal{P}_e(A_*)$ at $F = F_{A_*}$, and is zero elsewhere. The

[6] This same argument could also be raised against loss augmentation; however, loss augmentation is usually accomplished not by changing the planner, but rather by modifying the input cost function

gradient of (6.2.13) is

$$\nabla \mathcal{O}_F^{\mathcal{P}}[C_a] = \mathcal{P}_e(A_*^{\mathcal{P}})(\delta_F(F_{A_e}) \quad - \quad \delta_F(F_{A_*^{\mathcal{P}}}))$$

This gradient equals $\mathcal{P}_e(A_*^{\mathcal{P}})$ at $F = F_{A_e}$, $-\mathcal{P}_e(A_*^{\mathcal{P}})$ at $F = F_{A_*^{\mathcal{P}}}$, and is zero elsewhere. Therefore $\langle \mathcal{O}_F, \mathcal{O}_F^{\mathcal{P}} \rangle$ equals $\mathcal{P}_e(A_*)\mathcal{P}_e(A_*^{\mathcal{P}})$ if $A_* \neq A_*^{\mathcal{P}}$ and $2\mathcal{P}_e(A_*)\mathcal{P}_e(A_*^{\mathcal{P}})$ otherwise. Since $\mathcal{P}_e$ is defined to be greater than or equal to zero, the two gradients are always correlated (as long as they are nonzero).

□

What is implied by theorems 6.2.2 and 6.2.3 is that the two objectives share a common purpose: to make A_e as inexpensive as possible. While the two different optimizations may also be raising the cost on different actions at the same time, they will never work directly at cross purposes, and will not be satisfied until they reach the same cost function. Therefore, minimizing the objective in (6.2.14) will seek the same solution[7] as (6.2.13), which is bounded as specified in theorem 6.2.1. Therefore, when it comes to an actual implementation, (6.2.14) can be used so that the planner in question need not be modified.

In terms of actually defining $\mathcal{P}_e$, The obvious first choice would simply be to set $\mathcal{P}_e = L_e$. However, this would not account for errors that result in low loss but still produce undesirable behavior. It would also unnecessarily further penalize errors that result in high loss, but actually produce reasonable behavior. Instead, $\mathcal{P}_e$ should be chosen to complement L_e. The common issue with the examples presented in Figure 6.4 is that regardless of the lack or presence of immediate errors, the future behavior of the robot does not match the demonstrated behavior. Fundamentally, there is no way to guarantee such a match, as a single arc planner simply does not look far enough into the future. However, if a small error in a single action eventually results in a trajectory very different from that which was demonstrated, it implies that it might be more important to get that example right. The converse is also true; if a large error in a single action does not result in a drastically different trajectory, then it is not as important to get that example right.

Therefore, it is proposed to add a penalty function based on the simulation of future behavior. Define $\mathcal{T}^{C_a,a,x,g}$ as the actual trajectory that the robot will follow under cost function C_a from state x to goal g, when the first action undertaken is a. $\mathcal{T}^{C_a,a,x,g}$ is computed by repeatedly querying the planner to choose an action, simulating the effect of that action over a single period of the planning cycle, and then repeating the process. By setting $a = A_*$, the future behavior when executing A_* and then operating under the current cost function C_a can be approximated. The penalty function $\mathcal{P}_e(A_*, C_a)$ can be defined as an error function between $\mathcal{T}^{C_a,a,x,g}$ and the example behavior P_e. In addition to A_*, C_a is now taken as input to the penalty function (to indicate the explicit dependence on the current cost function). The specific form of $\mathcal{P}_e$ used in this work is further discussed in Section 6.3.1.

Along with helping to ensure that important examples are properly learned, slack re-scaling has another beneficial effect: it ensures that unimportant examples are not weighted too heavily. This is especially important when learning from potentially noisy and imperfect expert demonstration. That is, human drivers do not necessarily follow critically damped control laws, and demonstrated behavior may contain transients or biases that would otherwise lower the quality of the learned

[7] When multiple examples or timesteps are present, there is no explicit guarantee of converging to the exact same solution

behavior[8]. In other cases, the expert demonstration may be correct, but the specific scenario may be such that a wide range of actions will result in almost identical behavior; in such a case the learner should still try and learn the example action, but not at the expense of other, more important cases.

Using this new definition of $\mathcal{P}_e$ and reintroducing dynamics to (6.2.14) yields an objective functional of

$$\begin{aligned}\mathcal{O}[C_a] = \lambda\textsc{Reg}(C_a) \; &+ \sum_t \mathcal{P}_e(A^t_*, C_a)\left(C_a(F^t_{A^t_e}) + \sum_{s \in T^{A^t_e, x^t, g, t}} C_s(F^t_s)\right) \\ &- \sum_t \mathcal{P}_e(A^t_*, C_a)\left((C_a(F^t_{A^t_*}) - L^t_e(A^t_*)) + \sum_{s \in T^{A^t_*, x^t, g, t}} C_s(F^t_s)\right)\end{aligned} \tag{6.2.15}$$

Application of the product rule and collecting terms yields a new functional gradient of

$$\begin{aligned}\nabla\mathcal{O}_F[C_a] = &\sum_t \mathcal{P}_e(A^t_*, C_a)\left(\delta_F(F^t_{A^t_e}) \; - \; \delta_F(F^t_{A^t_*})\right) \\ &+ \sum_t \mathcal{P}'_e(A^t_*, C_a)\left(\mathcal{C}_e - \mathcal{C}_* - L^t_e(A^t_*)\right)\end{aligned} \tag{6.2.16}$$

The first component of the gradient is essentially the same as it has been previously, simply weighted by the value of the penalty function. However, the second term is more problematic, as it involves the gradient of the penalty function with respect to the cost function. For a penalty function based on the simulation of future behavior,this term implies that one way to lower the penalty (and by extension penalty weighted objective), is to modify the cost function to affect future behavior. For example, if the example action is a hard left, and the current planned action a hard right, then this term suggests to modify the cost function such that the action at time $t + 1$ is a lateral movement left plus a perhaps slight rotation. More generally, this term would seek to lower the cost of arcs that result in the future trajectory being closer to the demonstrated trajectory, and vice versa.

The gradient of $\mathcal{P}_e$ can not be explicitly defined without first defining $\mathcal{P}_e$ itself. However, it can be defined for generic $\mathcal{P}_e$ as

$$\begin{aligned}\mathcal{P}'_e(A^t_*, C_a) &= \sum_{(x^\tau, A^\tau_i) \in \mathbb{B}} \delta_F(F^\tau_{A^\tau_i}) \; - \sum_{(x^\tau, A^\tau_i) \in \mathbb{W}} \delta_F(F^\tau_{A^\tau_i}) \quad \tau \geq t \\ \mathbb{B} &= \{(x^\tau, A^\tau_i) \mid x^\tau \in \mathcal{T}^{C_a, A^t_*, x^t, g} \wedge \mathcal{P}_e(A^\tau_i, C_a) < \mathcal{P}_e(A^\tau_*, C_a)\} \\ \mathbb{W} &= \{(x^\tau, A^\tau_i) \mid x^\tau \in \mathcal{T}^{C_a, A^t_*, x^t, g} \wedge \mathcal{P}_e(A^\tau_i, C_a) \geq \mathcal{P}_e(A^\tau_*, C_a)\}\end{aligned} \tag{6.2.17}$$

For each state along the simulation of future behavior ($x^\tau \in \mathcal{T}^{C_a, A^t_*, x^t, g}$), $\mathcal{P}'_e$ differentiates between the actions that would result in lowering the penalty value (relative to the penalty under the current cost function) and the actions that would result in raising the penalty value.

However, there are obvious problems with using this new term during learning. For instance,

[8] The same issue led to the development of R-LEARCH for terrain preferences

in the example above imagine that the example action was a hard left to avoid a large obstacle, and the planned action a hard right to avoid the same obstacle. Once a direction has been chosen, the robot needs to commit to that to avoid hitting the obstacle. However, following the gradient term implied by $\mathcal{P}_e'$ would involve lowering the cost of all the left turn arcs, even those that result in a collision with the obstacle. More fundamentally, the problem is that the expert demonstration involves specific action choices for a specific time t and state x^t. The $\mathcal{P}_e'$ term attempts to raise or lower the cost of specific actions at time $\tau \geq t$ and state x^τ. However, there is no reason to believe that any of the choices $\mathcal{P}_e'$ prefers are actually what the expert would have done if confronted with x^τ. More formally, none of the choices in either $\mathbb{B}$ or $\mathbb{W}$ were actually made by an expert; therefore there is no reason to prefer or avoid them relative to A_*^τ

Fortunately, there is reason to ignore the $\mathcal{P}_e'$ term in the actual optimization.

Theorem 6.2.4. *Define*

$$\begin{aligned}\nabla\mathcal{O}_F^t[C_a] =& \mathcal{G}_L + \mathcal{G}_P\\ \mathcal{G}_L =& \mathcal{P}_e(A_*^t, C_a)\left(\delta_F(F_{A_e^t}^t) \;-\; \delta_F(F_{A_*^t}^t)\right)\\ \mathcal{G}_P =& \mathcal{P}_e'(A_*^t, C_a)\left(\mathcal{C}_e - \mathcal{C}_* - L_e^t(A_*^t))\right)\end{aligned}$$

Performing gradient descent by only following $\mathcal{G}_L$ will produce a C that meets the MMP constraint at time t while also minimizing $\textsc{Reg}(C)$

Proof. By definition, $\mathcal{G}_L(F_{A_e^t}^t) = \mathcal{P}_e(A_*^t, C_a)$ and $\mathcal{G}_L(F_{A_*^t}^t) = -\mathcal{P}_e(A_*^t, C_a)$. $\mathcal{G}_L = 0$ for any other value of F.

From (6.2.17) $(x^t, A_e^t) \in \mathbb{B}$ and $(x^t, A_*^t) \in \mathbb{W}$. Thus, $\mathcal{G}_P$ can be split into two components, one that accounts for these two feature values and another that accounts for all other feature values. Define

$$\begin{aligned}\mathcal{G}_P =& \mathcal{G}_P^c + \mathcal{G}_P^f \qquad (6.2.18)\\ \mathcal{G}_P^c =& \left(\mathcal{C}_e - \mathcal{C}_* - L_e^t(A_*^t))\right)\left(\delta_F(F_{A_e^t}^t) \;-\; \delta_F(F_{A_*^t}^t)\right)\\ \mathcal{G}_P^f =& \left(\mathcal{C}_e - \mathcal{C}_* - L_e^t(A_*^t))\right)\left[\sum_{(x^\tau, A_i^\tau)\in\mathbb{B}/(x^t, A_e^t)} \delta_F(F_{A_i^\tau}^\tau) \;-\; \sum_{(x^\tau, A_i^\tau)\in\mathbb{W}/(x^t, A_*^t)} \delta_F(F_{A_i^\tau}^\tau)\right] \quad \tau \geq t\end{aligned}$$

$\mathcal{G}_P^c$ is just a scaled version of $\mathcal{G}_L$, and seeks to lower the penalty by making the current example action the preferred action. $\mathcal{G}_P^f$ represents the rest of $\mathcal{P}_e'(A_*^t, C_a)$ which seeks to lower the penalty by making the future trajectory closer to the example one. Since $\mathcal{G}_L$ is only nonzero at two specific values of F, and $\mathcal{G}_P^f$ is explicitly zero at those two values, we can redefine the gradient as

$$\begin{aligned}\nabla\mathcal{O}_F^t[C_a] =& K\mathcal{G}_L + \mathcal{G}_P^f \qquad (6.2.19)\\ \langle\mathcal{G}_L, \mathcal{G}_P^f\rangle =& 0\end{aligned}$$

As the gradient consists of two orthogonal components, a reasonable optimization strategy would be to follow one component until it is zero, and then follow the other component until it is zero (essentially a special case of conjugate gradient). However since $\mathcal{G}_L$ is just a scaled version of the normal LEARCH gradient, we know from (5.1.11) that $\mathcal{G}_L = \vec{0}$ when the MMP constraint is satisfied

$$\mathcal{G}_L = \vec{0} \Longleftrightarrow A_*^t = A_e^t \Longleftrightarrow L_e = 0 \Longleftrightarrow P_e = 0 \Longrightarrow \mathcal{G}_P^f = \vec{0}$$

However, the final implication is not two way. That is, future behavior could be made equivalent to demonstrated behavior *except for* the original incorrect action at time t, resulting in $\mathcal{G}_P^f = \vec{0}$ while $L_e, P_e > 0$, requiring[9] further optimization by following $\mathcal{G}_L$

Therefore, the most regularized way to meet the slack-rescaled constraints in (6.2.9) is to ignore $\mathcal{G}_P$ and follow $\mathcal{G}_L$.

□

More intuitively, $\mathcal{G}_L$ tries to drive L_e and P_e to zero by matching behavior at time t. $\mathcal{G}_P^f$ tries to minimize P_e by minimizing future error. However, this alone will not necessarily drive L_e and P_e to zero. Since $\mathcal{G}_L$ is provably sufficient (Theorem 5.1.1 and Equation (5.1.11)), complicating the gradient will result in a solution that is farther from C_0 (with potentially worse generalization). This of course raises the question of what effect does slack rescaling actually have in this context. The answer is that it affects the balancing of multiple constraints (terms in the optimization). The optimization still seeks to achieve zero loss (and by consequence zero penalty) for each term. However, when this cannot be achieved it will give more weight to certain terms than others (based on the associated penalties). Therefore, in this work the $\mathcal{G}_P$ terms of the functional gradient in (6.2.16) are simply ignored. Section 6.4 provides empirical evidence that this optimization still results in reductions of the penalty terms.

The resulting algorithm is known as Penalty Weighted Dynamic LEARCH (PD-LEARCH) and is shown in Algorithm 7. PD-LEARCH is quite similar to Algorithm 6; the primary difference is the simulation of future behavior, and then weighting by the associated penalty. The need to simulate future behavior at each timestep of each example can significantly increase the computational burden. Fortunately, this extra step is only necessary when $A_*^{t,i} \neq A_e^{t,i}$; in practice this results in early iterations being more computationally expensive, with fewer simulations performed in later iterations of the algorithm. Another caveat is that the overall optimization is now nonconvex (as the inter-example weights can change between gradient steps). In practice, this simply requires a little more care with the selection of the learning rate (such as a fast decay, and a bit of trial and error for the initial value). In general, PD-LEARCH is applicable to any receding horizon system where the planner does not actually plan all the way to its goal, and small errors can propagate over time.

[9] It is possible that when projecting the gradient onto a direction set, following $\mathcal{G}_P^f$ will produce zero penalty and loss (through generalization of the cost function); it is simply not ensured

```
Algorithm 7: PD-LEARCH for planner preference models with known perception cost
Inputs : Example Behaviors $P_e^1, P_e^2, ..., P_e^n$, Sensor Histories $\mathcal{H}^1, \mathcal{H}^2, ..., \mathcal{H}^n$, Perception
         Cost Function $C_s$
$C_{a_0} = 1$;
foreach $P_e^i$ do
    for $\tau =$firstTime($P_e^i$) : $\Delta\tau$ :lastTime($P_e^i$) do
        $P_e^{\tau,i} =$ extractPathSegment($P_e^i$,$\tau$,lastTime($P_e^i$));
        $A_e^{\tau,i} =$ extractFirstAction($P_e^i$,$\tau$);
        [$\mathcal{F}_s^{\tau,i}$, $\mathcal{V}^{\tau,i}$] = simulatePerception($\mathcal{H}^i$,firstTime($P_e^i$),$\tau$);
for $j = 1...K$ do
    $T_f = T_o = T_w = \emptyset$;
    foreach $P_e^{t,i}$ do
        $\mathcal{M}^{t,i} =$ buildCostmap($C_s$,$\mathcal{F}_s^{t,i}$,$\mathcal{V}^{t,i}$);
        $A_*^{t,i} =$ chooseLossAugAction(start($P_e^{t,i}$),goal($P_e^{t,i}$),$\mathcal{M}^{t,i}$, $C_{a_{j-1}}$);
        if $A_*^{t,i} \neq A_e^{t,i}$ then
            $\mathcal{F}_a^{t,i,e} =$ computeActionFeatures($A_e^{t,i}$, $\mathcal{M}^{t,i}$);
            $\mathcal{F}_a^{t,i,*} =$ computeActionFeatures($A_*^{t,i}$, $\mathcal{M}^{t,i}$);
            $\mathcal{T}^{t,i} =$ simFutureBehavior($C_{a_{j-1}}$, $A_*^{t,i}$,start($P_e^{t,i}$),goal($P_e^{t,i}$));
            $\mathcal{P} =$ trajectoryError($P_e^{t,i}$, $\mathcal{T}^{t,i}$);
            $T_f = T_f \bigcup \mathcal{F}_a^{t,i,e} \bigcup \mathcal{F}_a^{t,i,*}$;
            $T_o = T_o \bigcup -1 \bigcup 1$;
            $T_w = T_w \bigcup \mathcal{P} \bigcup \mathcal{P}$;
    $R_j =$ trainWeightedRegressor($T_f$, $T_o$, $T_w$);
    $C_{a_j} = C_{a_{j-1}} * e^{\eta_j R_j}$;
return $C_{a_K}$
```

6.2.4 Simultaneously Learning Planner and Terrain Preference Models

Combining the dual objective in (6.1.2) with the penalty weighted objective of (6.2.15) yields a new combined objective of

$$\begin{aligned}\mathcal{O}[C_s, C_a] = {} & \lambda_s \textsc{Reg}(C_s) + \lambda_a \textsc{Reg}(C_a) \\ & + \sum_t \mathcal{P}_e(A_*^t, C_a)\left(C_a(F_{A_e^t}^t) + \sum_{s \in T^{A_e^t, x^t, g, t}} C_s(F_s^t)\right) \\ & - \sum_t \mathcal{P}_e(A_*^t, C_a)\left((C_a(F_{A_*^t}^t) - L_e^t(A_*^t)) + \sum_{s \in T^{A_*^t, x^t, g, t}} (C_s(F_s^t) - L_e^t(s))\right)\end{aligned} \tag{6.2.20}$$

The partial derivative of this objective with respect to C_a is shown in (6.2.16), and results in the PD-LEARCH algorithm. The previous section held C_s constant; however, if C_s is now also considered unknown, the partial derivative can be defined as

$$\nabla \mathcal{O}_F[C_s] = \sum_t \left[\mathcal{P}_e(A_*^t, C_a) \left(\sum_{s \in T^{A_e^t, x^t, g, t}} \delta_F(F_s^t) \;-\; \sum_{s \in T^{A_*^t, x^t, g, t}} \delta_F(F_s^t) \right) + \mathcal{G}_P^t \right] \tag{6.2.21}$$

If as before the $\mathcal{G}_P$ term is ignored, this gradient looks very similar to (5.2.4). The two key differences remaining are the additional weighting by the penalty term, and the summation over $T^{A_*^t, x^t, g, t}$ or $T^{A_e^t, x^t, g, t}$ instead of P_* or P_e respectively. This latter difference simply implies that the expert demonstration should be modified to match a trajectory that the planning system is capable of; this modification was already mentioned for perception cost functions with constrained planners (Section 5.4.3) and is described further in the next section. Therefore, this gradient implies that along with using PD-LEARCH to learn a planner cost function, PD-LEARCH can also be applied to the task of learning perception cost functions.

Practically speaking, as opposed to using PD-LEARCH for learning perception cost functions, DR-LEARCH should be used instead. The reason is simply the different definitions of the loss function: as the perception loss function is defined over the states each trajectory traverse, it will have a form very similar to the penalty function[10]; therefore this extra weighting is unnecessary in the perception domain. In addition, the robust extensions developed for DR-LEARCH were shown (Section 5.5) to significantly improve performance, and therefore should still be applied.

Therefore, PD-LEARCH can be used to learn C_a when C_s is considered constant, and DR-LEARCH can be used to learn C_s when C_a is considered constant. Given that these two cost functions are (by design) independent, and that they make use of the same input training data (A set of example trajectories P_e), it is possible to learn both functions at the same time in a single optimization procedure. At each iteration, an update can be computed for each cost function (the computation of which will make use of the current value of the 'other' cost function), and this overall procedure repeated until convergence of both functions. This algorithm is shown in Algorithm 8, which combines D-LEARCH (Algorithm 3) and PD-LEARCH (Algorithm 7). The full DR-LEARCH implementation (Algorithm 4) is withheld solely for the sake of clarity.

6.3 Application To Mobile Robotic Systems

An important issue that was raised in Section 5.4.3 is that of the action/plan space available to both the expert and the planning system. In the case of learning a cost function over constant curvature arcs, an expert example exists in the uncountably infinite range $[-\text{MAXCURV}, \text{MAXCURV}]$, while the planner only has $|\mathcal{A}|$ choices available. This prevents the optimization from having a proper termination condition, as it is unlikely that A_* will ever exactly equal A_e except at a boundary. As seen in Section 5.4.3 this can result in overlearning specific examples, and reduce generalization. The simple solution to this issue is rather than using the exact A_e demonstrated by the expert, choose the closest arc in $\mathcal{A}$ (with respect to curvature) as the new example.

In Chapter 5 this same solution was applied, with the caveat that the expert's example action was not known reliably (and could not be taken as the expert's actual intended action); instead a heuristic procedure was used to identify the most likely action. Such an approach is not necessary

[10] this is not the case for the planner loss function, and is why penalty weighting was applied in the first place

Algorithm 8: Learning perception and planner preference models

```
Inputs : Example Behaviors $P_e^1, P_e^2, ..., P_e^n$, Sensor Histories $\mathcal{H}^1, \mathcal{H}^2, ..., \mathcal{H}^n$
$C_{s_0} = 1$;
$C_{a_0} = 1$;
foreach $P_e^i$ do
    for $\tau =$firstTime($P_e^i$): $\Delta\tau$ :lastTime($P_e^i$) do
        $P_e^{\tau,i} =$ extractPathSegment($P_e^i$,$\tau$,lastTime($P_e^i$));
        $A_e^{\tau,i} =$ extractFirstAction($P_e^i$,$\tau$);
        $[\mathcal{F}_s^{\tau,i}, \mathcal{V}^{\tau,i}] =$ simulatePerception($\mathcal{H}^i$,firstTime($P_e^i$),$\tau$);

for $j = 1...K$ do
    $U +_= U_- = \vec{0}$;
    $T_f^a = T_o^a = T_w^a = \emptyset$;
    foreach $P_e^{t,i}$ do
        $\mathcal{M}^{t,i} =$ buildCostmap($C_{s_{j-1}}$,$\mathcal{F}_s^{t,i}$,$\mathcal{V}^{t,i}$);
        $P_*^{t,i} =$ planLossAugPath(start($P_e^{t,i}$),goal($P_e^{t,i}$),$\mathcal{M}^{t,i}$);
        $A_*^{t,i} =$ chooseLossAugAction(start($P_e^{t,i}$),goal($P_e^{t,i}$),$\mathcal{M}^{t,i}$, $C_{a_{j-1}}$);
        foreach $x \in P_e^{t,i} \bigcap \mathcal{V}^{t,i}$ do
            $U_-(\mathcal{F}_x^{t,i}) = U_-(\mathcal{F}_x^{t,i}) + 1$;
        foreach $x \in P_*^{t,i} \bigcap \mathcal{V}^{t,i}$ do
            $U_+(\mathcal{F}_x^{t,i}) = U_+(\mathcal{F}_x^{t,i}) + 1$;
        if $A_*^{t,i} \neq A_e^{t,i}$ then
            $\mathcal{F}_a^{t,i,e} =$ computeActionFeatures($A_e^{t,i}$, $\mathcal{M}^{t,i}$);
            $\mathcal{F}_a^{t,i,*} =$ computeActionFeatures($A_*^{t,i}$, $\mathcal{M}^{t,i}$);
            $\mathcal{T}^{t,i} =$ simFutureBehavior($C_{a_{j-1}}$, $A_*^{t,i}$,start($P_e^{t,i}$),goal($P_e^{t,i}$));
            $\mathcal{P} =$ trajectoryError($P_e^{t,i}$, $\mathcal{T}^{t,i}$);
            $T_f^a \ = T_f^a \bigcup \mathcal{F}_a^{t,i,e} \bigcup \mathcal{F}_a^{t,i,*}$;
            $T_o^a \ = T_o^a \ \bigcup -1 \quad \bigcup 1$;
            $T_w^a \ = T_w^a \bigcup \mathcal{P} \quad \bigcup \mathcal{P}$;
    $T_f^s = T_o^s = T_w^s = \emptyset$;
    $U = U_+ - U_-$;
    foreach $\mathcal{F}_s^{t,i}$ such that $U(\mathcal{F}_s^{t,i}) \neq 0$ do
        $T_f^s \ = T_f^s \bigcup \mathcal{F}_s^{t,i}$;
        $T_o^s \ = T_o^s \bigcup$ sgn($U(\mathcal{F}_s^{t,i})$);
        $T_w^s = T_w^s \bigcup |U(\mathcal{F}_s^{t,i})|$;
    $R_j^s =$ trainWeightedRegressor($T_f^s$,$T_o^s$,$T_w^s$);
    $C_{s_j} = C_{s_{j-1}} * e^{\eta_j^s R_j^s}$;
    $R_j^a =$ trainWeightedRegressor($T_f^a$,$T_o^a$,$T_w^a$);
    $C_{a_j} = C_{a_{j-1}} * e^{\eta_j^a R_j^a}$;
return $C_{s_K}, C_{a_K}$
```

in this case with respect to A_e; however, it is still necessary to bootstrap the remainder of the example trajectory. This is because P_e can be any path from the start to the goal, while the planned behavior under consideration at each iteration, $T^{A^t_*,x^t,g,t}$, is much more restricted: for the first couple of meters (the arc length) there are only $|\mathcal{A}|$ possible choices. The solution (as indicated by Equations (6.2.8) and (6.2.21) is to use $T^{A^t_e,x^t,g,t}$ instead of P_e. When C_s is known a priori, the definition of $T^{A^t_e,x^t,g,t}$ is concrete; however, when C_s itself is being modified, the question naturally arises of what C_s is used to generate $T^{A^t_e,x^t,g,t}$. The solution to this issue is the same as applied in Section 5.4.3. First, a perception cost function C^g_s is learned using an unconstrained (global) path planner. Then, $T^{A^t_e,x^t,g,t}$ is defined as the path resulting from following arc A_e, and then the optimal path (under C^g_s) from the end of the arc to the goal.

6.3.1 Penalty Function Design

$\mathcal{P}_e$ was previously defined as an error function between the expert behavior P_e and $\mathcal{T}^{C_a,a,x,g}$, the simulation of future behavior under the current cost function. However, this formulation can result in a high penalty despite low loss between A_e and A_*, if the trajectories diverge not at time t, but at time $t+k$. In such a case, choosing the correct action at time t is made to seem more important than it actually is. The larger the value of k, the lower the likelihood that a high penalty has anything to do with the decision made at time t

A solution to this problem is to only simulate future behavior out do a certain time horizon. However, if the simulated and demonstrated behavior are significantly different (a situation which would call for a high penalty) a time horizon may not allow the simulated behavior to sufficiently diverge. An alternative is to use a distance horizon, cutting off the simulation once the trajectory leaves a circle of radius r_{sim} from the starting point. Therefore, instead of defining $\mathcal{P}_e$ as en error function between $\mathcal{T}^{C_a,a,x,g}$ and P_e, this work uses an error function over $\mathcal{T}^{C_a,a,x,g,r_{sim}}$ and $P_e^{r_{sim}}$, where both trajectories are truncated outside accordingly.

The issue now is to define the actual error function between $\mathcal{T}^{C_a,a,x,g,r_{sim}}$ and $P_e^{r_{sim}}$. Equation (5.5.1) provides a starting point for such a function, based on the 2D locations of points of equal arc length along two trajectories. In general, such a function provides an excellent template for $\mathcal{P}_e$. That is, if the simulated future behavior and actual example behavior are quite similar, the penalty should be low, and high if the trajectories diverge (if the trajectories initially diverge but then re-converge, the penalty is intermediate). However, there are certain explicitly bad cases that this error function would not capture. A common example of such a case is if the planner follows a trajectory that generally tracks the example, but with the vehicle oriented in the opposite direction. Another such case is if the planner actually becomes stuck: that is, it continues to oscillate and chooses actions that do not make progress towards its goal (such a case can only occur with a small planning horizon). Examples of both of these cases can be seen in Figure 6.6. Fortunately, these cases can generally be concretely defined and detected, and the penalty made high when they are triggered. Therefore, a penalty function of the following form is used in this work, producing output in the [0,1] range

$$\begin{aligned}
\mathcal{P}_e(A_*, C_a) &= \max(\hat{\mathcal{P}}_e(A_*, C_a), \mathcal{H}_e(A_*, C_a)) \qquad (6.3.1)\\
\hat{\mathcal{P}}_e(A_*, C_a) &= \frac{1}{|\mathcal{T}^{C_a, A_*}|} \sum_{x_e, x_* \in \mathbb{P}} [1 - \exp \|x_e - x_*\|^2/\sigma^2)]\\
\mathbb{P} &= \textsc{findCorrespondingPointPairs}(P_e, \mathcal{T}^{C_a, A_*})\\
\mathcal{H}_e(A_*, C_a) &= \textsc{wrongDirecton}(P_e, \mathcal{T}^{C_a, A_*}) \bigvee \textsc{stuck}(\mathcal{T}^{C_a, A_*}) \bigvee ...
\end{aligned}$$

$\hat{\mathcal{P}}_e(A_*, C_a)$ computes an error similar to (5.5.1); the difference is that instead of finding corresponding points based on location, they are found based on the traversed path length up to that point (This corrects for the case when the simulated trajectory eventually follows the example behavior, but performs some additional actions or oscillations first). $\mathcal{H}_e$ simply seeks to identify the aforementioned bad scenarios (as well as any additional ones that may apply), and ensure a high penalty in such cases.

There is of course some irony in the necessity to design a good penalty function in order to learn a good cost function, as it can devolve into the same sort of parameter tuning that this approach was supposed to mitigate. However this problem is far less complex than the original problem, and has the advantage of a known basis for comparison. That is, it is much easier to define what is clearly undesirable in comparison to known desirable behavior as opposed to trying to define what is desirable in the first place, let alone a hierarchy of desirability. Since the penalty function is in comparison to known desirable behavior, all that is really necessary is that it highly penalizes the cases that result in such undesirable behavior (and aren't already captured by the loss function. Therefore, in practice the exact form of the penalty function (and the exact values it produces) do not have as significant effect as one would expect, as long as in general the right errors are penalized.

6.3.2 Planner Feature Design

Computing meaningful and useful features is a core part of any perception system, and is well understood in that context. When features are computed for use in a preference model, there is often a well understood relationship between the feature and cost (e.g. object density). Even when this relationship is not as clear (e.g. color or other appearance information) it is well understood what types of features are useful. The situation is not quite as clear when constructing features for a planner preference model. One simple reason for this confusion is that explicit planner preference models are less common than perception preference models; as a result features are not always explictly computed. Even when they are, the relationship between the features and cost is rarely as intuitive as with perception.

Nevertheless, there are some common feature classes that can be applied. When the action set consists of constant curvature arcs, then the curvature of each arc is an obvious feature. When applicable, the vehicle steering angle provides a slightly transformed version of the same basic feature. The velocity of a particular action is also a very meaningful feature. When the motion planner does not have direct control of velocity, then the sign of velocity (i.e. forward or reverse) is still very important. Finally, in the case of the single arc motion planner, a notion of heading error is of extreme importance. There are many different ways to compute such a feature; the commonality is that they all capture the difference between the robot's heading along or at the end of an arc, and the heading that would be necessary to track the global path from the end of said arc. In such a way, heading error features allow the planner to prefer actions that align the vehicle with the global path in the forward direction (or the reverse direction if desired).

In addition to computing features from a specific action, features can also be computed that depend on the current state (in one sense, heading errors already fit this description). If the state vector is augmented to contain the robot's current linear and angular velocity (that is, the velocities at the time the next action is to be executed) then features can be computed from these values. For example, along with features based on the curvature of the specified action, features can also be computed based on the change in curvature between the robot's last and current action. Whether or not the robot is changing direction also proves to be a very important feature.

Once a feature set has been designed and implemented, additional features can be learned during training. The same feature learning phase as described in Section 5.4.1 can be easily implemented for planning as well as perception in Algorithm 8. As before, the primary advantage is to allow the use of linear cost functions (for computationally reasons as well as generalization) along with certain nonlinear feature combinations that prove useful. In the case of planner features, there are often specific thresholds that can prove to be quite useful in feature design; these thresholds can then be learned from demonstration rather than set and tuned by hand.

6.4 Experimental Results

6.4.1 Learning Planner Preference Models

A single arc motion planner was implemented adhering to (6.2.2) and using Field D* as the associated global planner. The planner's action set consisted of 21 forward arcs and 9 reverse arcs (Figure 6.3). Along with the planner, a simulator was implemented to allow the planning system's actual behavior to be observed. Obstacles of specified cost could also be added to the simulation environment. The simulator therefore provided a scenario where the perception cost function was known a priori, and only the planner cost function was unknown (as in Section 6.2). Training and validation examples were collected by having an expert drive the robot in the simulation environment. Collection of these sets required less than 2 hours of expert time.

This planning system was designed and implemented to match Crusher's local planner as closely as possible. However, there was one key difference. Crusher's local planner essentially had a two tiered preference model. Individual arcs were penalized or rewarded based on features of said arc. However, there was also a higher level state machine that monitored Crusher's state and progress over several planning cycles. The states in this machine each corresponded to different penalties at the arc level, and resulted in different abstract behavior goals (e.g. normal operation, explicit turning to align with the D* path, more deliberation motion in constrained areas, etc.). The use of such a tiered approach greatly improved the effectiveness of Crusher's planning system, allowing it to make complex maneuvers such as 3 point turns that were not explicitly in its action set. However, this performance came at the cost of additional complexity, and another time consuming hand tuning problem.

In contrast, the arc planner used for these experiments did not have such a state machine; a

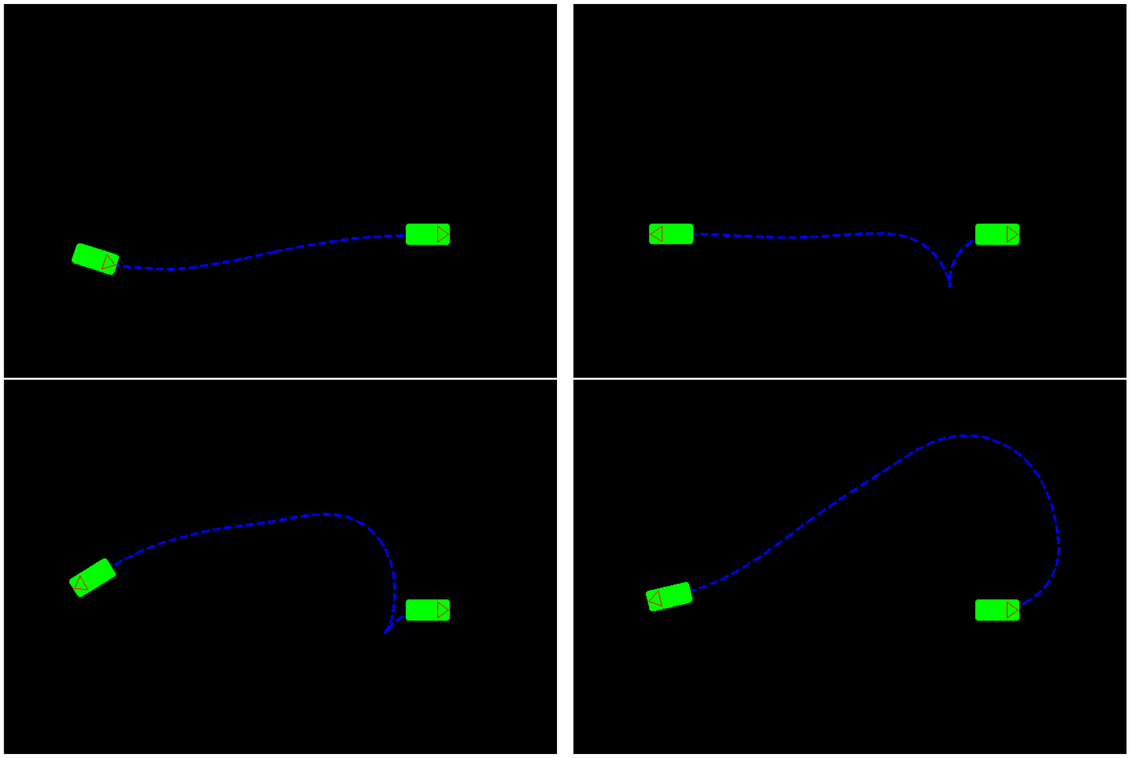


Figure 6.5: The progression of learned preference models (from top left to bottom right) to match the scenario in Figure 6.1. Over time, the planner learns to prefer forward to reverse more and more, as well as align itself with the heuristic (D*) path.

single preference model was learned and used for all actions in the arc set. Despite this limitation, the preference model was capable of generating similar complex behavior (see Figures 6.5, 6.6 and 6.7). This is because all the necessary information for these maneuvers is in the features. Therefore, Crusher's local planning system is another example of a case where added complexity and or decreased potential was caused by the necessity of human parameter tuning[11].

Figure 6.5 shows the evolution of the robot's behavior in the simulator, as the cost function evolves, with a goal mirroring that in Figure 6.1. The desired behavior in such a case (based on various examples in the training set) was for the robot to drive forward all the way to the goal. Initially, the robot simply chooses actions that get it to the goal as quickly as possible, resulting in it driving in reverse all the way. However, very quickly a preference for being oriented forward along the D* path to goal (based on heading features) emerges. This preference results in the robot performing a V-turn style maneuver to orient itself, then driving forward to the goal. As the preference model continues to evolve, a preference for driving forwards grows stronger, resulting in shorter V-turns. Eventually, this preference grows strong enough that the V-turn is eliminated; instead the robot makes a hard turn and drives forwards all the way to the goal.

Figure 6.6 shows a more complex test scenario, with the robot starting in a cul-de-sac. Initially, the robot again simply chooses actions that minimize the heuristic cost, driving in reverse out of

[11] That is, Crusher's state machine was implemented not because it made the system more capable, but because it made the system easier to tune. In cases where such state machines do actually add to the core planning capability, the conditions for state transitions (as well as behavior within a state) could be learned by PD-LEARCH.

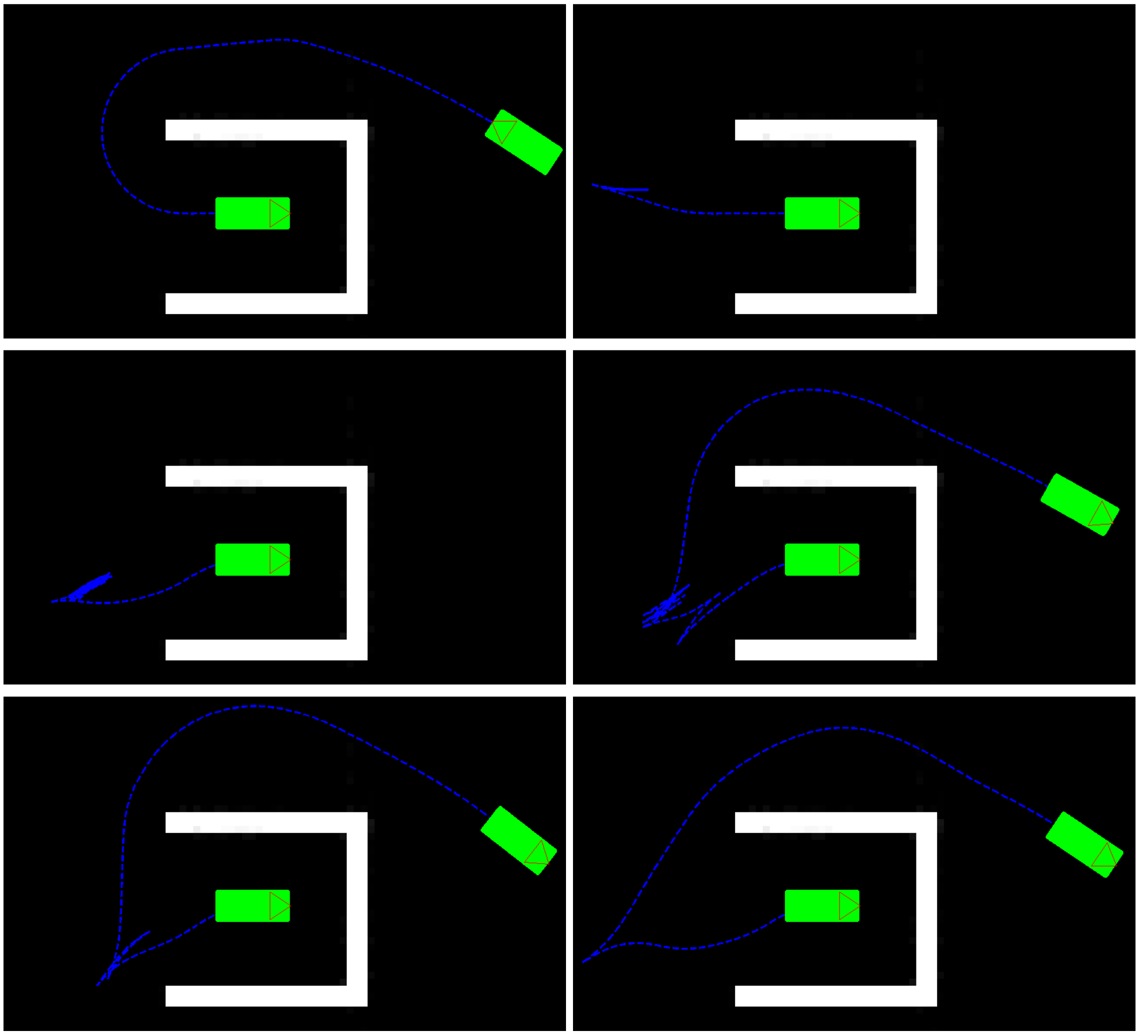


Figure 6.6: The progression of learned preference models (from top left to bottom right) in a validation scenario. Over time the planner learns to exit the cul-de-sac to align itself with the D* path to the goal. In intermediate iterations, it does not turn far enough, and becomes stuck in oscillatory behavior (never reaching the goal). In later iterations, this oscillation is reduced and finally eliminated.

the cul-de-sac and all the way to the goal. Again, a preference for being oriented with the D* path emerges. At first, this results in a straight, back up. However, this puts the robot in a bad state: it reverses until it has a little room to turn forward, but not enough room. The result is an oscillatory behavior, and the robot becomes permanently stuck. As the preference model continues to evolve, the initial turn becomes tighter, and is followed for longer: eventually, it evolves to the point where the robot is able to escape the oscillation, and drive forward to the goal. However, learning is not finished: it continues until the amount of actual oscillation is first reduced, and eventually eliminated. The ability to generalize preferences in such a way that complicated maneuvers evolve out of short horizon planning, without an explicit state machine or mode switching, demonstrates the overall effectiveness of this approach.

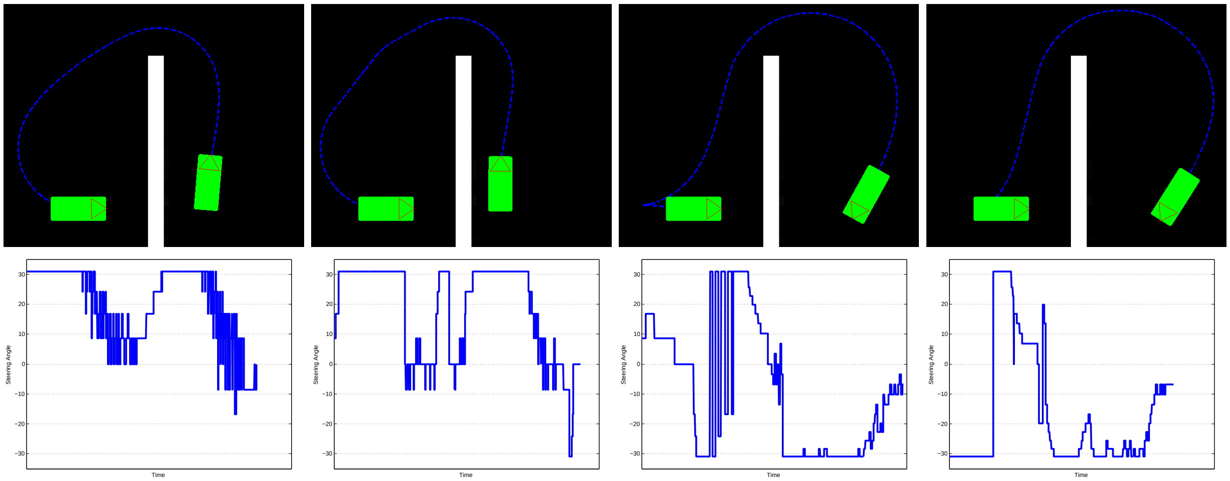
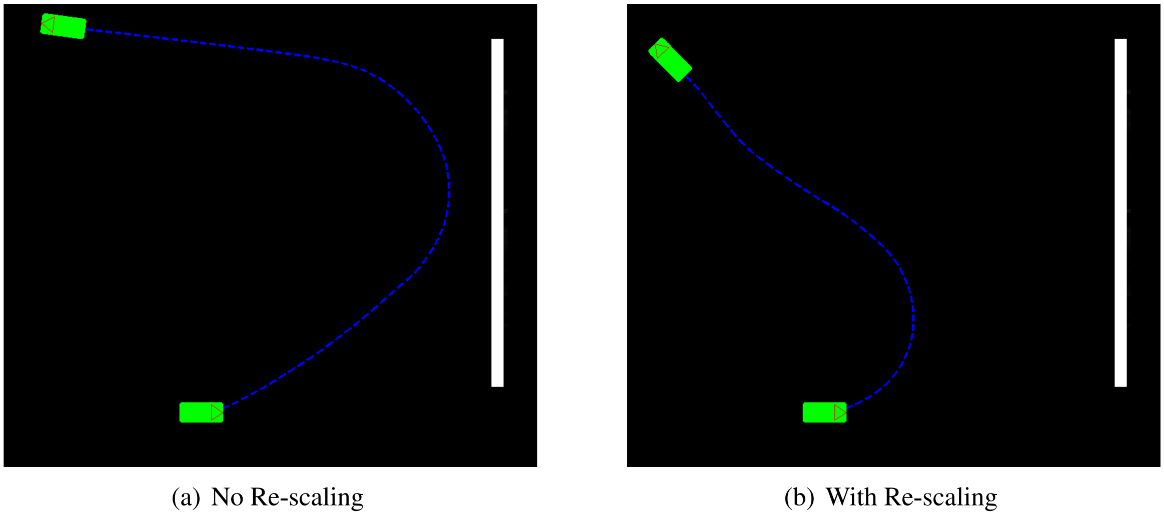


Figure 6.7: The progression of learned preference models (from left to right) in a validation scenario. The commanded curvature at each point in time is also shown for example example. Over time the planner first learns to avoid unnecessary turning, and then to favor driving forward and being aligned with the D* path.

Figure 6.7 shows a similar evolution of complex behavior, in this case backing up before driving forward around a wall. Along with the trajectories at different stages of learning, the commanded curvature over time is shown for each example. Initially, there is no preference to avoid constant changes in curvature, resulting in significant oscillations in the chosen actions. Over time, these oscillations are damped; the planner has learned to match the expert's tradeoffs between avoiding unnecessary turns while still turning to achieve a goal and avoid obstacles.

Generalization and the ability to perform such maneuvers is improved by the use of slack re-scaling in PD-LEARCH as described in Section 6.2.3. Figure 6.8 demonstrates a concrete example of this difference. In Figure 6.8(a), a preference model was learned without slack re-scaling. When operating under this model, the robot does not turn hard enough towards its goal, and actually drives further away; it only begins to turn hard when it has to in order to avoid an obstacle (otherwise it would never achieve its goal). Such behavior is obviously quite undesirable, and its non-intuitive as to why it is learned in the first place. The cause of this issue is the fact that the learning optimization (without re-scaling) is trying to balance numerous examples with equal weight. For a full example trajectory that turns hard towards a goal and maintains that turn, there are only a few timesteps that correspond to increasing curvature and then maintaining the turn at its hardest; there are far more that simply correspond to driving towards a goal. Because all timesteps are weighted equally, the optimization considers it just as important to get each one of those timesteps exactly right. Even if the optimization can get $n-1$ timesteps of a full example exactly correct, the single incorrect one may be the most important. Further, that single error may result in a robot state with similar features, resulting in the error being repeated (as in Figure 6.8(a), where the error of not turning hard enough is continually repeated).

In contrast, by implementing slack re-scaling and weighting examples by their associated simulation penalty, the optimization can give more credence to getting certain examples right; essentially, the most important examples are identified and emphasized. In contrast, the optimization may no longer get $n-1$ examples exactly right, but it will emphasize the important examples to



(a) No Re-scaling

(b) With Re-scaling

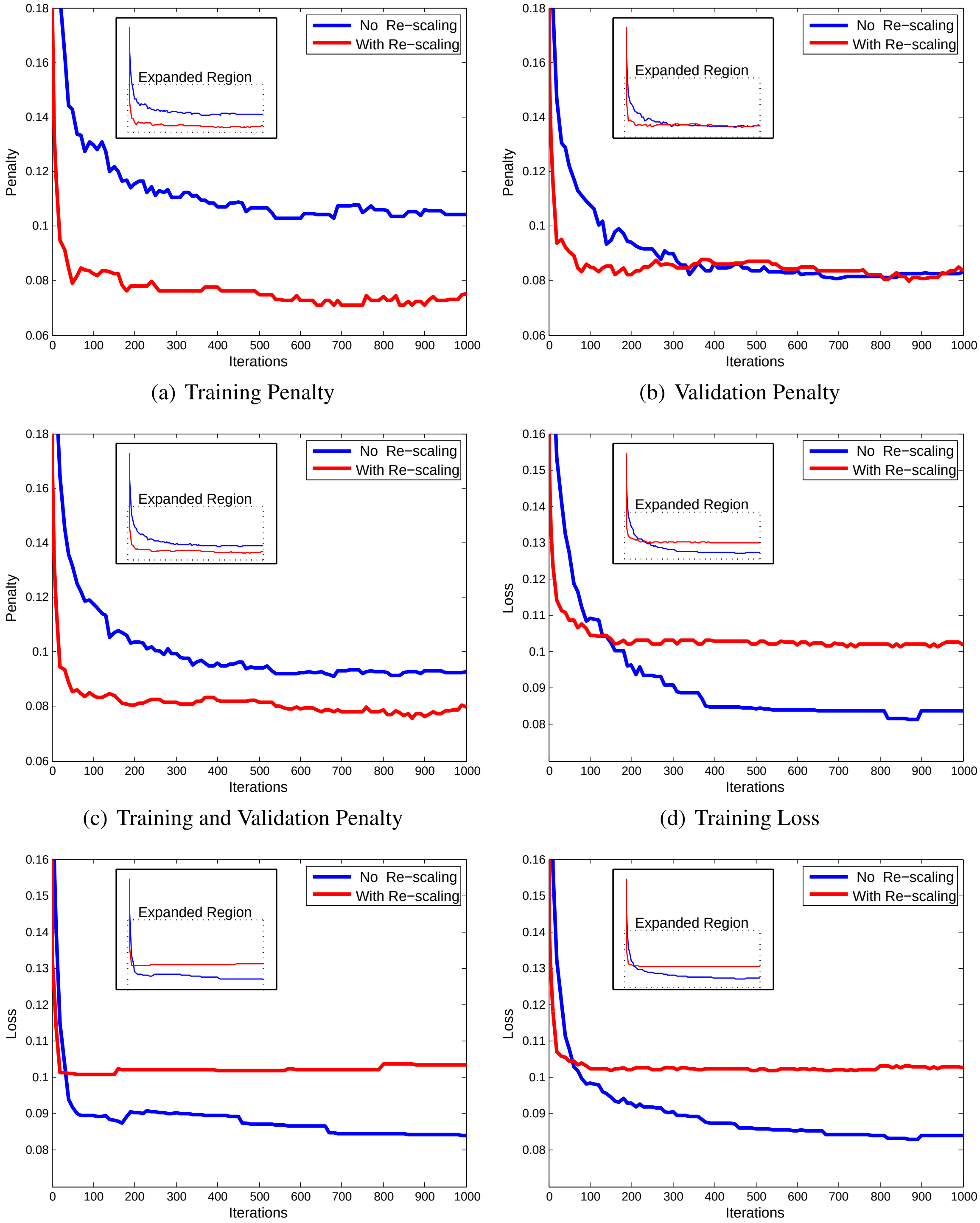
Figure 6.8: Example of robot behavior with and without slack re-scaling. Without it, the robot does not turn hard enough to ever reach its goal, only correcting to avoid collision with an obstacle.

matching expert behavior. Figure 6.8(b) demonstrates the same scenario with a preference model learned using slack re-scaling; the result is a hard turn towards the goal.

Figures 6.9 and 6.11 demonstrate quantitatively the difference when using slack re-scaling. Figure 6.9 shows the results of experiments to learn planner preference models based on 12 planner features. As would be expected, the average training penalty is significantly lower when using slack re-scaling; timesteps where immediate errors propagate are weighted more heavily in the optimization, and those where errors are of less future consequence are weighted less. However, this improvement does not carry over to the validation performance; due to a very small feature set, there is equally good generalization. Nevertheless, slack re-scaling is able to improve on the specific errors it sees. This improvement comes at the expense of the loss (defined in (6.2.6) as 1 if the action is the wrong direction, and the error in curvature otherwise). That is, although slack re-scaling does a better job of matching overall behavior, it does a poorer job of matching individual actions. This is both an expected and necessary tradeoff: weighting less important timesteps less will result in higher immediate errors, but those errors will not propagate.

Figure 6.10 presents these same results, along with the performance obtained when using DR-LEARCH instead of PD-LEARCH. When using DR-LEARCH for planner preference models, there is no slack re-scaling (and therefore no explicit bounding of the penalty). Instead, as in Section 5.3.2 the example action is replanned at each iteration; instead of using A_e^t, the lowest cost action whose curvature is within a small bound to A_e^t (and is also in the same direction) is used. This adds a degree of robustness to noisy demonstration; however it does not explicitly identify and increase the constribution of important examples. The result is an improvement in the average training penalty, but not as significant as with PD-LEARCH.

Figure 6.11 shows the results of the same set of experiments when using 48 planner features (these features were mostly non-linear versions and conjunctions of the original 12 features, added to improve performance with linear cost functions). The training penalty and loss follow the same pattern: slack re-scaling lowers the average penalty, while increasing the average loss. However, the validation performance follows a difference pattern. For both the average validation penalty



(a) Training Penalty

(b) Validation Penalty

(c) Training and Validation Penalty

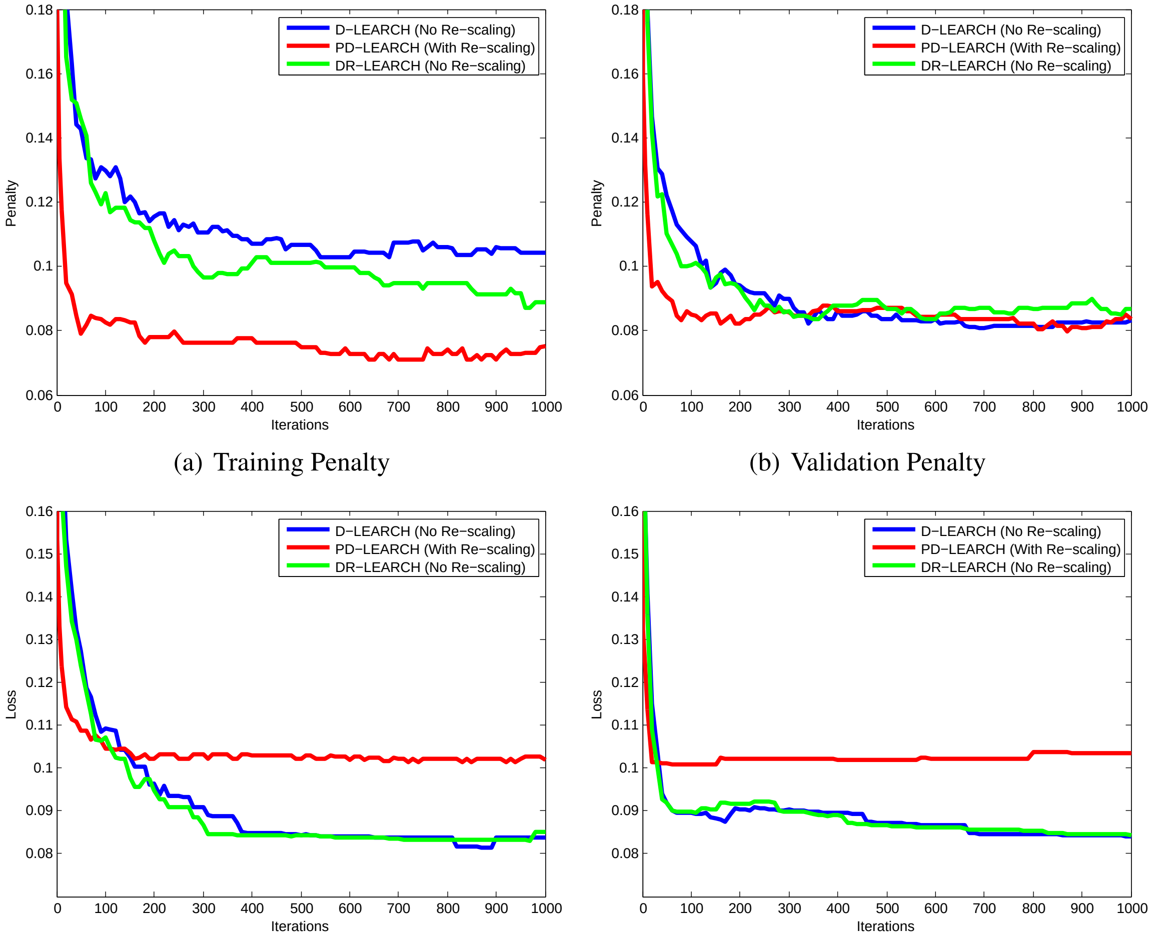
(d) Training Loss

(e) Validation Loss

(f) Training and Validation Loss

Figure 6.9: Learning planner preference models with 12 planner features

and loss, without slack re-scaling there is significant overfitting. This demonstrates the ability of slack re-scaling to deal with noisy and imperfect expert demonstration. By weighting individual examples by their consequences, slack re-scaling is able to essentially ignore instances where the



(a) Training Penalty

(b) Validation Penalty

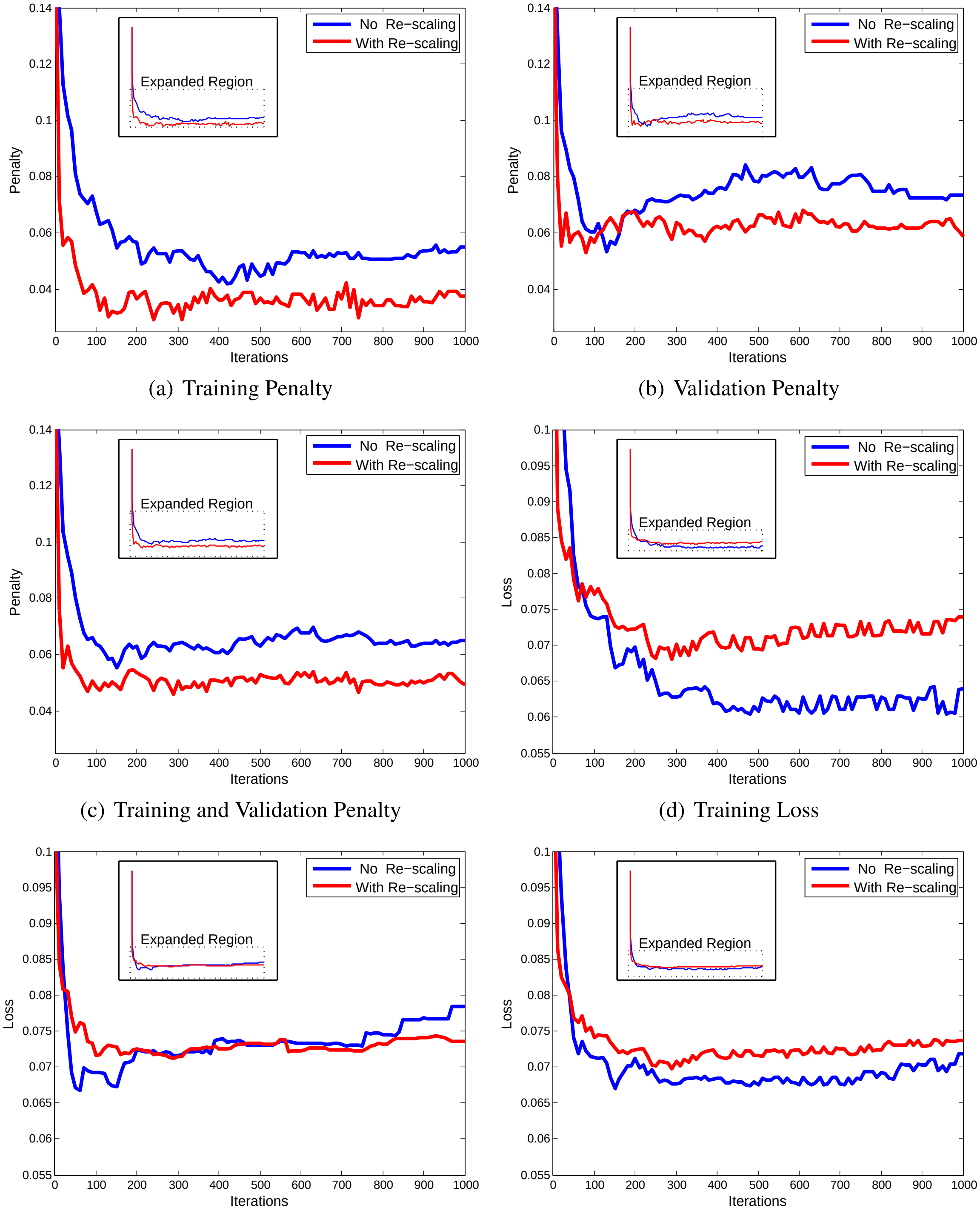
(c) Training Loss

(d) Validation Loss

Figure 6.10: Performance of PD-LEARCH vs DR-LEARCH for learning planner preference models.

example action is not exactly optimal (e.g. turning very slightly when the goal is straight ahead, or oversteering when making a hard turn), such as the case demonstrated in Figure 6.4(c). In contrast, without this robustness the optimization will continue to try to fit these small errors. When the feature space is small this may not affect generalization (as Figure 6.9 demonstrates). However, a higher dimensional feature space allows for overfitting and less generalization. Therefore, when operating off training data from human experts (even when said training data is collected under ideal conditions, such as in simulation) slack re-scaling (PD-LEARCH) is necessary to ensure robustness to noise.

In order to obtain a comparison of learned versus hand-tuned performance, an experiment was performed by having an independent expert attempt to hand tune a preference model for the planner while operating in the simulator. The expert was constrained to using a linear cost function (as with the learned function), but was allowed to add new features as he saw fit. Every time the simulator was started with a modified set of parameters, the current parameters were logged in order to



(a) Training Penalty

(b) Validation Penalty

(c) Training and Validation Penalty

(d) Training Loss

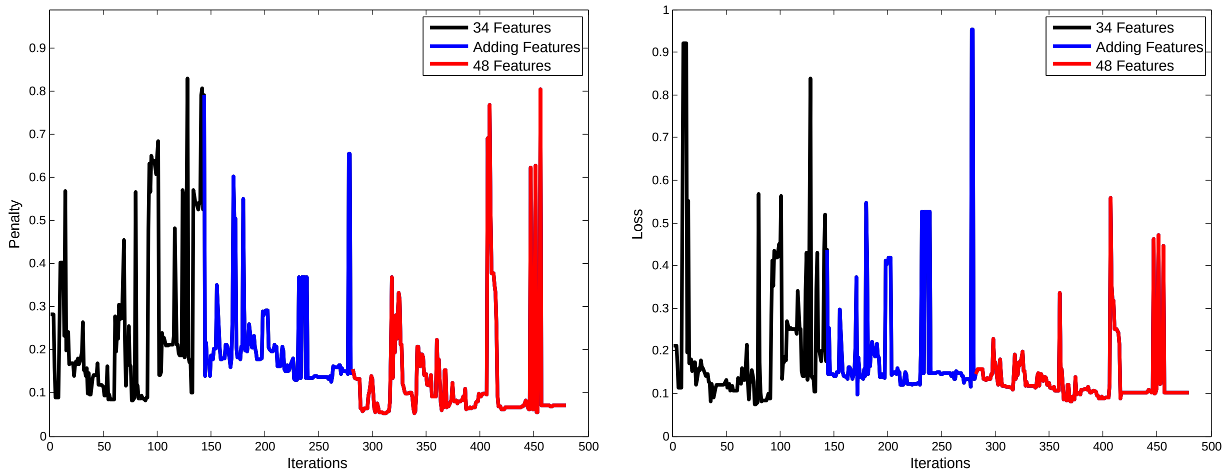
(e) Validation Loss

(f) Training and Validation Loss

Figure 6.11: Learning planner preference models with 48 planner features

evaluate performance after each tuning. The overall tuning procedure went through 3 phases [12]. Initially, there were 34 features and associated weights, which were tuned by the expert. Next,

[12] These phases were not mandated, but rather were simply the result of the expert's tuning procedure



(a) Validation Penalty (b) Validation Loss

Figure 6.12: Hand-tuning planner preference models. Tuning began with 34 features, and then an additional 14 were added during tuning by the expert.

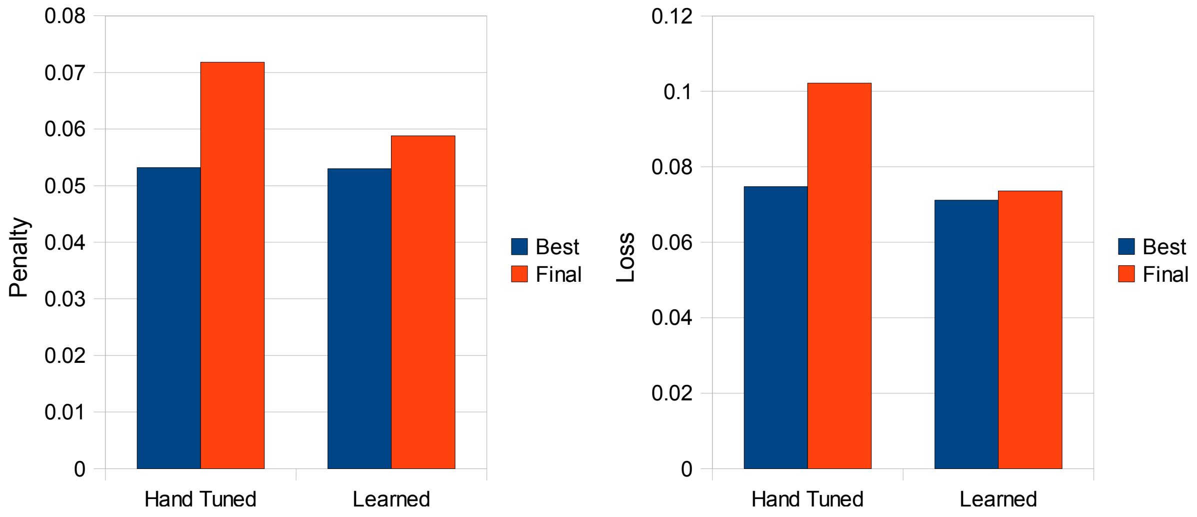


Figure 6.13: A comparison of the best versus the final validation performance achieved by each approach during tuning.

there was a period where additional features were slowly added and tuned, eventually resulting in 48 features (these 48 features were used in the experiment in Figure 6.11). Finally, there was a long period of tuning the parameters for the final 48 features. Overall, this process took approximately 12 hours of expert time, spread over 2 days.

Figure 6.12 presents the results of this experiment, showing the average validation penalty and loss at each parameter set. Unlike an automated tuning procedure (Figure 6.11), the hand tuning is very unstable. Good validation performance is achieved very quickly; however subsequent tunings decrease performance. When performance is decreased, it is not by some small percentage, but by very large amounts (the worst validation performance is 3-4 times worse than the initial performance). This demonstrates that although a human tuner may have intuition about the effect of tuning individual parameters, the appropriate scale is generally not know, and is only found by trial and error. Over time, the hand tuning procedure begins to approach a steady state floor, but there are still lots of spikes as large changes are made and then undone (if they do not improve performance in the expert's opinion).

Of note is the difference between the performance of the best parameter set during human tuning, and the final parameter set. Figure 6.13 presents these results, along with the comparable results for the learned tuning. The hand tuning achieves best case validation performance that compares quite well with the learning approach. However, there is a significant difference in the final performance. The hand tuning, after several spikes and restarts, settles on a parameter set that is significantly lower performance than the best case. In contrast, the final validation performance of the learned system is closer to its best case. This improved validation performance was accompanied by a dramatic reduction in the required expert interaction.

An important point to re-emphasize when comparing hand and automated tuning is that a human expert can only consider (and test against) a small number of scenarios at once. Therefore, hand tuning is prone to overfit whatever small set of scenarios the expert is currently focused on. In contrast, an automated approach considers a much larger set of examples, and checks against each one every time even the smallest change is made. This make it more likely that an automated system will catch a parameter change with a negative effect, and less likely that significant overfitting will occur. In addition, implementing an automated approach naturally provides the mechanism for performing automated validation on independent examples, providing a mechanism for early stopping (or reversion to earlier settings) when overfitting does occur. When parameter tuning is performed by hand, such a mechanism is almost never implemented[13].

6.4.2 Learning Perception and Planner Preference Models

The arc planner was ported to the E-Gator robotic platform (Figure 6.14) for field testing and experimentation. Along with this planning system, a subset of Crusher's perception system was ported to the E-Gator as well. Thus, the E-Gator autonomy system is conceptually quite similar to Crusher, and follows a similar data flow (Figure 5.9). Just as with Crusher, key to the performance of the E-Gator system are two preference models: a perception cost function to map features of 2D locations into terrain costs, and a planner cost function to indicate relative costs of specific actions.

This platform offered two key advantages for conducting experiments into the learning from demonstration approaches described in this work. First, the E-Gator is retrofitted to be a drive by wire vehicle; however it can also still be driven by a human driver. Therefore, expert demonstration can be provided by someone actual sitting in and driving the vehicle, as opposed to remote control. This is especially important for issues of driving style, as a remote operator would not be as concerned with (nor necessarily as capable of) demonstrating smooth, clean motion. The second advantage is that the E-Gator system did not have previously existing hand tuned preference models (as the planner and perception implementations were done in conjunction with this work). Therefore, just as with the simulator hand tuning experiment, hand tuning of the full E-Gator system (perception and planning) could be recorded in detail.

Although the E-Gator perception system was based on modules ported from the Crusher sys-

[13] That is, hand tuning could just as easily revert to an earlier, 'best' parameter setting as a learned tuning, but implementing the validation mechanism requires almost as much work as implementing full learning from demonsrtation.

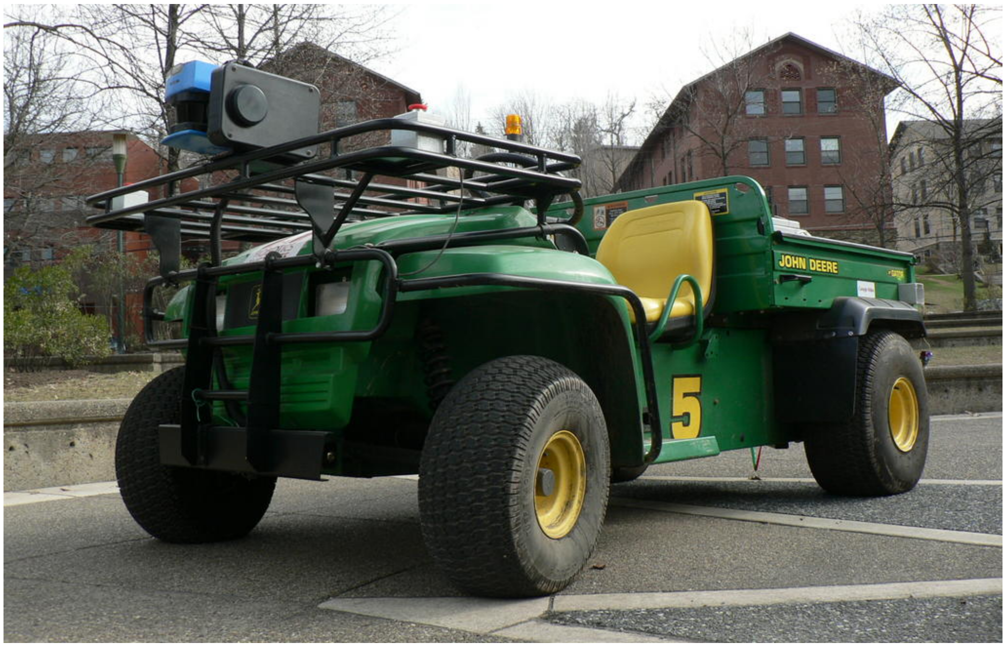
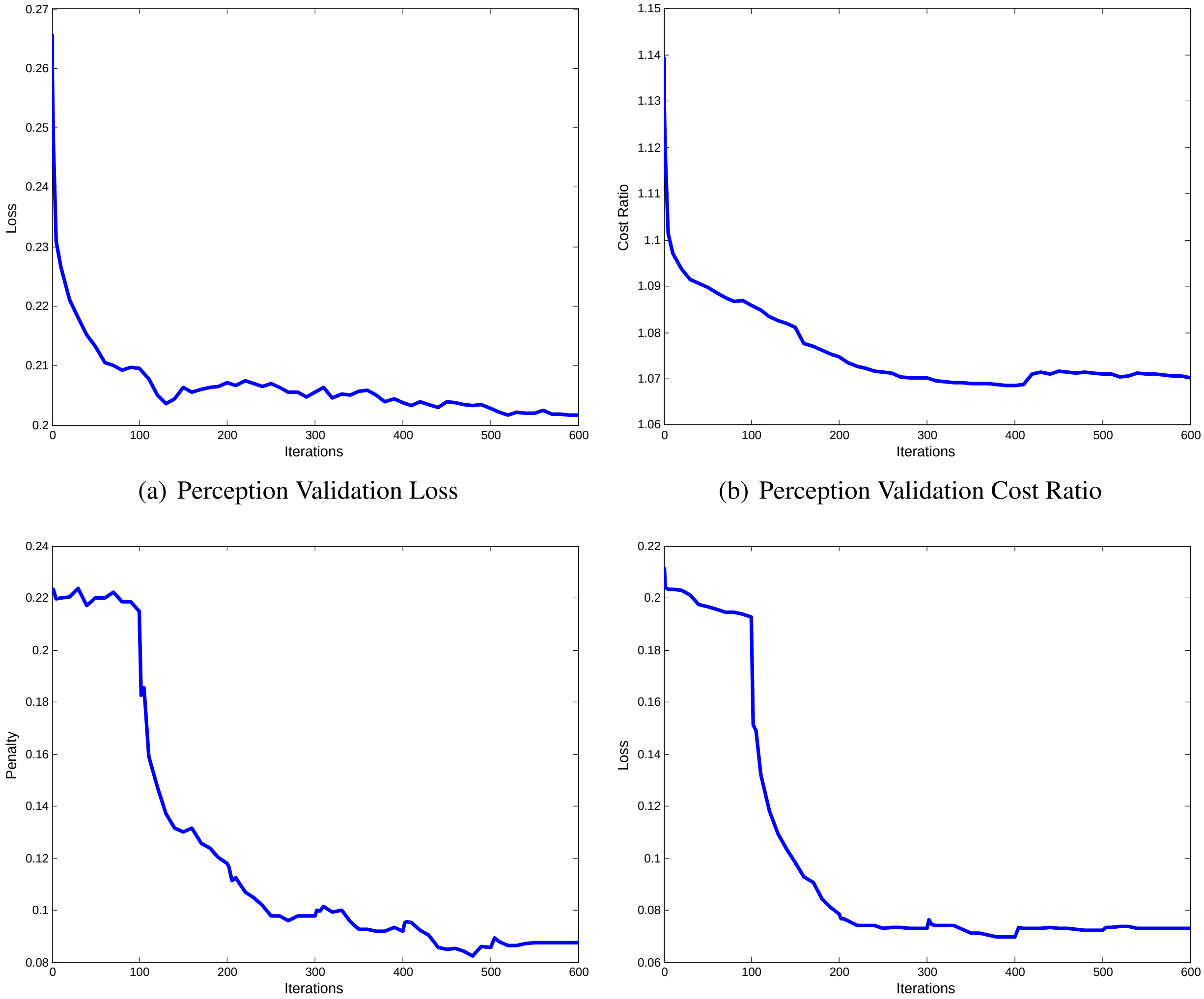


Figure 6.14: The E-Gator Robotic Platform

tem, there are a few key differences. The most important of these is the available sensors: The E-Gator has only a single nodding laser, with a 100° field of view. There is also a single camera; however it was not utilized for these experiments. There are also differences in the features produced by the two systems. Crusher's perception system produced a large set of geometric shape features; however these were not fed directly to the cost function. Instead, they were used as input to the terrain classifier, with classifications being fed to costing. On the E-Gator, no classifier was utilized. Instead, a subset of the shape features was fed directly to costing. Another difference was the addition of various smoothed features over varying windows. The purpose of such features was to allow the cost of a particular location to be affected not only by the terrain at that location, but by the terrain near it; as a result, a soft cspace expansion of cost maps was possible. Such a soft cspace expansion is often used to prevent a robot from getting too close to high cost areas when they can be avoided, while still allowing it to squeeze through tight areas when necessary.

The combined optimization for simultaneously learning perception and planning preference models (Algorithm 8 was implemented for the E-Gator system. This implementation simply fused the planner learning implementation from the simulator experiments with the implementation used for Crusher's perception system in Section 5.5 (including the robust extensions inherent in DR-LEARCH). Training and validation sets were collected by demonstrating and recording desired behavior on the E-Gator in the field. The demonstration was intended to imply both where the robot should drive, and how it should drive. The terrain varied from semi-structured, man-made environments (e.g. parking lots and around buildings) to more unstructured terrain (e.g. small and large trees, rolling hills, tall grass and bushes, etc.). In addition to data collected live on the E-Gator, the data sets used for planning were augmented by examples from the simulator. This allowed the easy addition of specific examples deemed important during training. The final planner preference model used 60 features (12 additional features to the 48 used in the last section) and 35 features for the perception preference model. Overall, collecting the training set required less than



(a) Perception Validation Loss

(b) Perception Validation Cost Ratio

(c) Planner Validation Penalty

(d) Planner Validation Loss

Figure 6.15: Learning perception and planner preference models for the E-Gator Robot

4 hours of human interaction (factoring in that two people are necessary to operate the E-Gator for safety reasons).

Figure 6.15 presents the validation results of the learning algorithm. In this context, evaluation of performance is complicated by the dependence between the two cost functions. That is, the performance of the planner preference model is dependent on having a good perception preference model, and vice versa. To remove this dependence, the perception system was instead evaluated based on performance of the global D* planner (which does not have a dependence on the planner cost function). That is, the error metrics were computed by comparing the global D* plan (with each cost function) to the expert demonstration. As described in Section 5.4.3 having the correct cost function for the global planner is a necessary but not sufficient condition for having the correct cost for the arc planner; however the difference in the final metrics is small enough to make it worthwhile to remove this dependence. Therefore, Figure 6.15 shows the validation loss and cost ratio for the perception cost function (with the global planner), and the validation penalty and loss for the planner cost function. All four metrics show a steady improvement during successive learning iterations, eventually converging to a steady state. Due to the non-convexity of the planner optimization, a small learning rate is used (for the planner) during the first 100 iterations; this accounts for the slow initial improvement. Overall, Figure 6.15 demonstrates the effectiveness of Algorithm 8 for learning coupled perception and planner preference models from a single set of expert demonstrations.

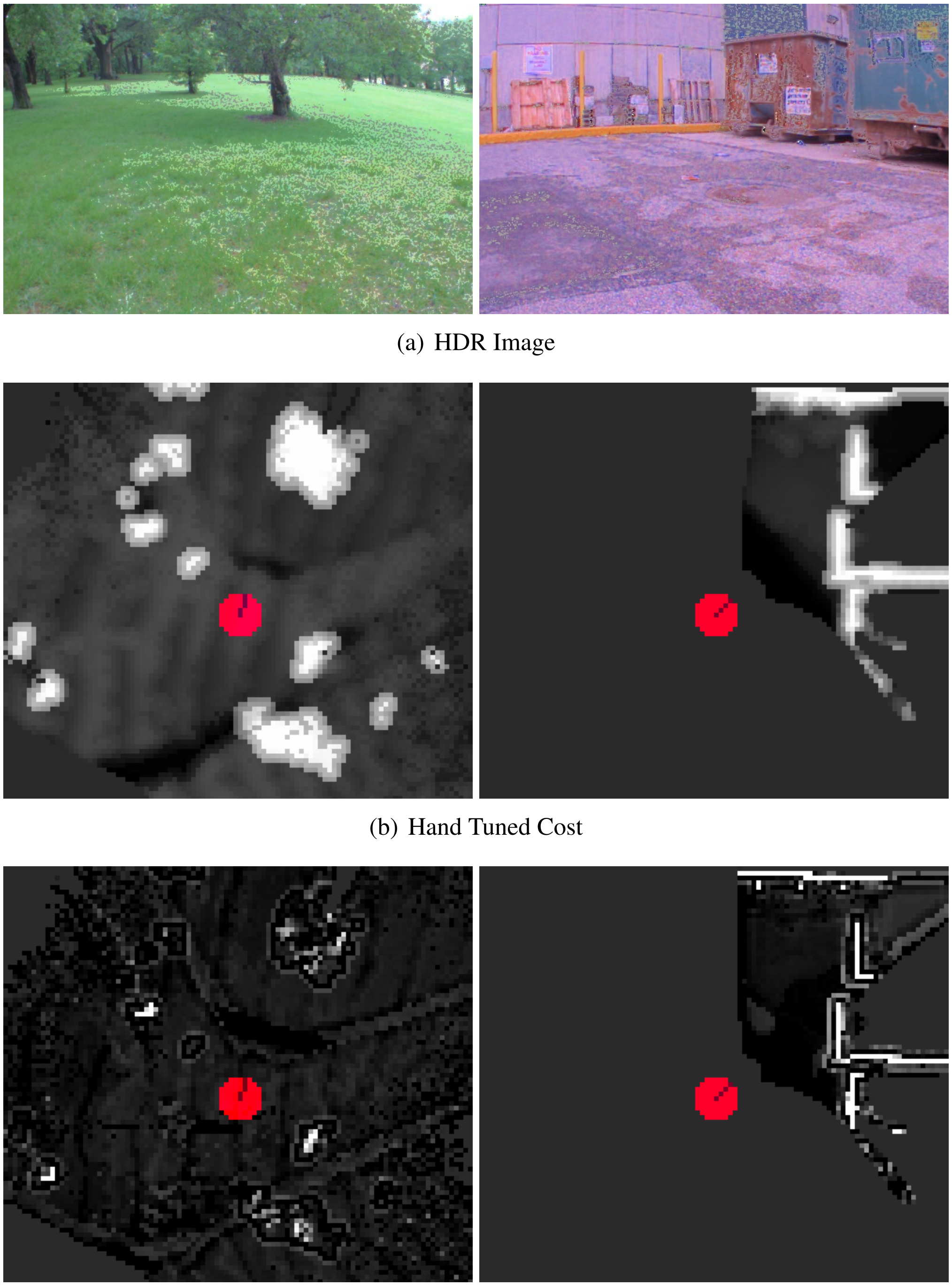
In order to compare learned versus hand tuned performance for the fielded E-Gator system, an independent robotics expert was tasked with tuning by hand both perception and planner cost functions. As in the similar experiment with the simulator, the state of both functions was recorded whenever parameters were modified, in order to evaluate performance over time. The expert began by tuning the planner in the simulator, and tuning perception by observing playback of a large set of data logs. Next, the E-Gator was taken out for a period of field testing, during which the robot's behavior was observed, and both preference models were tuned accordingly. Next, another period of offline tuning was performed. During this phase, many of the data logs used in tuning had been specifically collected to be examples of cases where the robot was currently having difficulty. Finally, the robot was taken out for another period of field testing, where final fine tuning was performed, until the expert was satisfied that no further significant progress was possible, and the experiment was declared over.

This back and forth between offline and online tuning of the system is very common for fielded mobile systems, and is quite similar to how Crusher was tuned over several years. Most of the actual tuning is done offline, due the difficulty and expense of field testing, and the ease of repeatable testing offline. However, since actual performance always diverges from simulated performance, robot field testing is necessary to tease out these differences, and then re-tune the system accordingly. Overall, the tuning of both preference models took 38 hours of tuning time, spread over 3 weeks. 18 of those 38 hours were spent tuning and testing the E-Gator in the field, which also required the presence of a safety operator, therefore bringing the total effort to 56 engineer-hours.

The final hand tuned perception cost function is shown for two representative scenes in Figure 6.16, along with the corresponding learned cost function. The differences in the cost function are quite similar to the differences between Crusher's hand tuned and learned cost functions (Section 5.5). The learned cost function appears noiser, as LEARCH anticipates the planners internal cspace expansion. Also of note is the soft expansion implemented in each cost function; it is a much sharper expansion in the learned system, which allows more precision maneuvers in tight spaces (when necessary). A final distinction is the differing cost on areas of tree canopy that the robot can drive under; the learned function distinguishes theres areas from tree trunks and lower branches; the hand tuned system makes no such distinction.

Figure 6.17 presents the quantitative results of this tuning experiment. As in Figure 6.15 the validation loss and cost ratio of the perception system (evaluated through the global planner) along with the validation penalty and loss of the planning system are presented. This tuning follows the same pattern seen in Figure 6.12; good performance is achieved fairly quickly, however subsequent tuning causes large spikes, as the expert attempts to fix specific observed problems. At times, the expert chooses to revert to a previous parameter set after exploring several significant changes. As the experiment progresses, performance begins to reach a steady state with fewer spikes; eventually parameter changes are minimized, until the final parameter set is reached.

Again of note is the difference between the performance the hand tuning procedure achieves in the best case, and the final performance when it has finished (Figure 6.18). For the perception



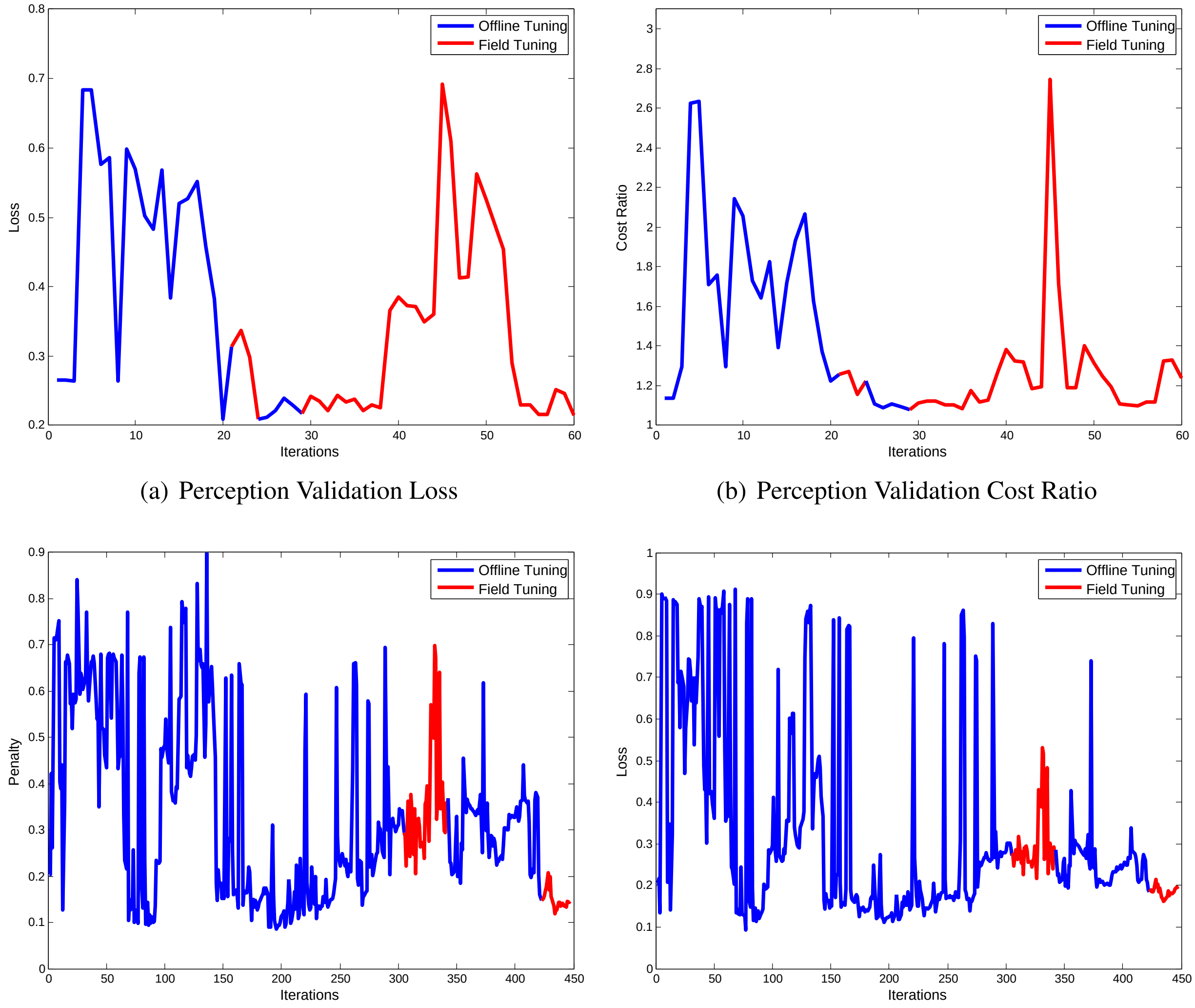
(a) HDR Image

(b) Hand Tuned Cost

(c) Learned Cost

Figure 6.16: Comparison of hand tuned vs learned cost functions in a unstructured and semi-structured scene.

system, the best validation performance achieved compares well with the best case performance of the learned system. However, it is again the case that the final hand tuned system achieves lower performance. While the increase in the loss is small, the significant increase in cost ratio implies that when the robot's planned path does not match the validation example, it is much farther



(a) Perception Validation Loss

(b) Perception Validation Cost Ratio

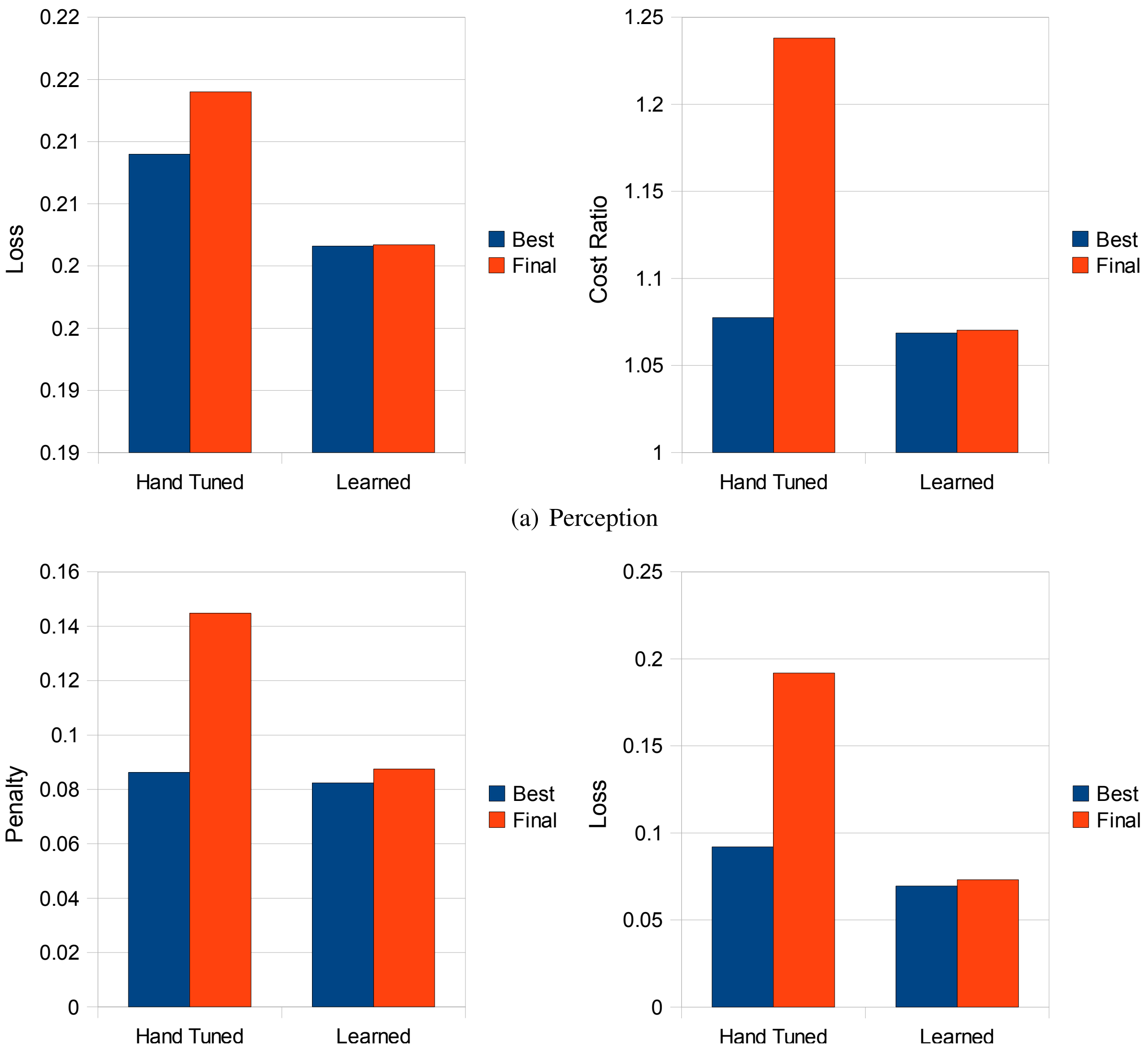
(c) Planner Validation Penalty

(d) Planner Validation Loss

Figure 6.17: Hand-tuning perception and planner preference models for the E-Gator Robot

from being correct. In contrast, the learned system's final performance is again similar to its best performance. The trend for the planning system is even more striking. Not only is the difference between best and final performance far larger for the planning system than the perception system, it is also more significant on the E-Gator as opposed to in the simulator (Figure 6.13). This speaks even more to the difficulty of validation when actual robot performance must be observed, and to the advantages of the automated validation that learning from demonstration provides.

Of course, the differences between simulated and actual robot performance that have just been emphasized also imply that the online performance of the learned and hand tuned preference models must be compared. To this end, a series of comparison experiments was conducted in the field with the E-Gator. These experiments were conducted in a similar manner to those involving Crusher described in Section 5.5.2. The E-Gator was run through a set of comparison courses, with each waypoint pair run twice: once with the learned preference models (perception and planning) and once with the hand tuned models. Waypoint spacing averaged just over 50m. Over the full



(a) Perception

(b) Planner

Figure 6.18: A comparison of the best versus the final validation performance achieved by each approach during E-Gator Tuning.

set of experiments, the E-Gator drove for more than 3 hours autonomously, traversing more than 4 km.

The results of these experiments are shown in Table 6.1, and present a very stark contrast in the behavior of the two systems. The most important difference is the rate of safety interventions necessitated by each pair of preference models: the hand tuned system had an intervention rate that was seven times that of the learned system. In addition to this explicit difference in safety, there were other significant differences in driving style. The learned system spent far less time in reverse (important for a vehicle with no rear sensors), and was also less prone to getting stuck in oscillatory back-and-forth behavior (as indicated by the higher variance in the number of direction switches of the hand tuned system). The variance in the distance traveled was also significantly higher in the hand tuned system, indicating that highly inefficient behavior was more likely. There was also

	Avg Dist (m)	Extra Dist (m)	Avg Roll ($\circ$)	Avg Pitch($\circ$)	% Time in Reverse
Learned (μ/σ^2)	69.27/785.5	12.02/148.5	3.71/2.61	4.45/2.29	0.075/0.012
Hand-Tuned (μ/σ^2)	73.09/1791.7	17.15/1293.24	4.24/3.90	4.39/2.15	0.19/0.079
P-value	0.34/**0.014**	0.23/**0.0**	0.13/0.14	0.56/0.57	**0.016/0.0**

	Dir Switch Per m	Avg Steer Angle ($\circ$)	Avg Δ Angle ($\circ$)	% Time Angle $\neq 0$	Avg Δ Angle $\neq 0$ ($\circ$)	Safety E-Stops
Learned (μ/σ^2)	0.029/0.0014	9.43/15.46	1.69/0.20	0.33/0.0064	5.19/0.49	0.13/0.12
Hand-Tuned (μ/σ^2)	0.044/0.0079	11.74/14.43	2.08/0.76	0.19/0.0033	10.76/3.60	0.90/1.49
P-value	0.18/**0.0**	**0.011**/0.57	**0.016/0.0**	**0.0/0.038**	**0.0/0.0**	**0.0/0.0**

Table 6.1: Results of E-Gator experiments comparing learned to hand-tuned performance.

a stark contrast in how the two systems steered. On average, the learned system turned less than the hand tuned system, and also turned the wheel less on average. This second difference comes despite the fact that the learned system kept the steering wheel steady a much smaller percentage of the time than the hand tuned system. The learned system simply made more, smaller movements of the steering wheel, while the hand tuned system made fewer, large movements of the steering wheel. On average, when the wheel was turned, it was turned twice as much by the hand tuned system as the learned system.

Despite this lowered reluctance to hard turns, most of the extra safety interventions came when the hand tuned system bumped or collided with significant obstacles such as trees. Qualitatively speaking, these collisions were usually the result of the planning system. That is, the perception cost always appeared sufficiently high to force the global planner around the obstacles, and appeared sufficiently high to account for the kinematic constraints of the E-Gator as well. The E-Gator would usually start to turn to avoid such obstacles, but then would sometimes begin to turn around them too soon, either clipping them with the back of the vehicle, or hitting them head on with a front wheel or bumper. It is interesting to note that this suboptimal behavior was somewhat expected, in that similar incidents were observed during hand tuning. In fact, a good deal of the planner parameter tuning (and consequently the spikes seen in later tuning iterations in Figure 6.17) was done in a direct attempt to correct this particular issue. However, the expert was unable to tune the planner to avoid such behavior while at the same time not introducing other significant negative behaviors (specifically crippling oscillations), and eventually settled on the final parameter set as a compromise. Therefore, the final results of this set of experiments strongly support the theory that manual parameter tuning, especially in high dimensions, is likely to be unable to produce reasonable results in a reasonable length of time.

Chapter 7

Stable, Consistent, and Efficient Learning from Expert Demonstration

The previous chapters have presented techniques that can automate the process of generating preference models through the collection of specific forms of expert demonstration. These approaches can eliminate the task of manual parameter tuning and model design; instead of an expert tuning the system to produce desired behavior, the system can learn to produce desired behavior from demonstration. However, it remains the responsibility of the expert to determine what behavior is desirable, and what demonstration to provide.

Looking forward to the possibility of general application of such approaches, there are some high level requirements one could imagine setting on the results

Stability The behavior of the robotic system after learning should not be too dependent on any single, key demonstration. That is, there should never be a key concept that is only taught by one example. Likewise, if a system has already been trained to have good performance, the addition of new training data should never reduce performance in any significant manner.

Consistency Learning from demonstration offers the promise of train once, tune forever; if individual perception, planning, or control systems are modified, preference models can simply be relearned from previous demonstration. However more significant changes such as the addition of sensors or the porting of existing software to a new robot will still require collection of new training data. In such instances, retraining multiple times from scratch should not result in systems with drastically different performance (taking into account whatever modifications forced the retraining).

Efficiency The collection of training data should remain an easy process (relative to manual parameter tuning), and should not grow to require unreasonable amounts of expert interaction, as this would negate a key advantage of this approach. In addition to minimizing the amount of expert interaction, the burden placed on the expert should also be minimized; that is the act of providing demonstration and feedback should be made as easy as possible.

Unfortunately, as is often the case these requirements are at odds. Stability is partially ensured by the robustness to noisy and imperfect demo provided by DR-LEARCH and PD-LEARCH. However, it also generally requires sufficient training data to ensure that all important concepts are learned, and that no single, very poor example can counteract them. Consistency also requires large datasets, to ensure not only that all important concepts are covered, but potentially multiple times; small, disjoint training sets may imply slightly difference preferences due to human inconsistency. In contrast, by definition efficient use of expert interaction requires as little training data as possible.

This chapter presents solutions aimed at decoupling these requirements. These solutions fall into two classes. The first class seeks different methods for the collection and interpretation of expert demonstration and feedback. By increasing the ease to an expert of providing demonstration and feedback, larger training sets can be collected in less time. Additionally, by clarifying exactly what it is that an expert means to teach (that is, ensuring that it is clear what concept the expert is implying) the natural inconsistency in human demonstration can be lessened in its effect. The second class of approaches make learning from demonstration more interactive, and seek to ensure that additional demonstration provided by an expert is useful and not redundant. This not only limits the amount of required training data, but can help insure stability and consistency by forcing the inclusion of any key examples that might exist.

7.1 Different Forms of Expert Demonstration and Feedback

The MMP framework and subsequent algorithms interpret an expert demonstration P_e from start s_e to goal g_e as implying the optimal trajectory between endpoints. This is a very powerful assumption. On one hand, it allows the inference of a vast number of constraints from each demonstration, which in turn makes each example very meaningful. On the other hand, it also requires the assumption that the expert demonstration is indeed optimal for some cost function. As shown in both of the previous chapters, this is rarely the case, as human demonstration is almost always noisy or suboptimal. Therefore, some of the constraints enforced by presumed optimality are incorrect. Robust extensions compensate for this issue; however, it would be beneficial to simply avoid it when possible. In addition, this presumption of optimality puts a burden on the expert, who must now ensure that provided demonstration is not only desirable, but near optimal. At times, the expert may not feel confident in making such a strong statement, but could still provide useful demonstration or feedback.

7.1.1 Relative Examples

It need not be the case that an expert demonstration be treated as an optimal demonstration.. The original MMP constraint (rewritten for general cost functions) is

$$\begin{gathered}\text{minimize } \mathcal{O}_F[C] = \textsc{Reg}(C) \qquad (7.1.1)\\ \text{subject to the constraints}\\ \sum_{x \in \hat{P}} (C(F_x)) \ \geq \ \sum_{x \in P_e} (C(F_x))\\ \forall \hat{P} \ s.t.\ \hat{P} \neq P_e, \ \ \hat{s} = s_e, \ \ \hat{g} = g_e\end{gathered}$$

The optimality of P_e is implied in a pairwise relative manner; that is, P_e should be lower cost than $\hat{P}$ for any $\hat{P}$. It is 'for any $\hat{P}$' that can cause problems if P_e is a desirable but suboptimal path.

A way to lessen the impact of this problem is to reduce the scope of what is implied by P_e. That is, instead of stating that P_e is better than any $\hat{P}$, it can be simply stated that P_e is better than a *specific* alternate path P_a

$$\text{minimize} \;\; \mathcal{O}_F[C] = \text{REG}(C) \qquad (7.1.2)$$
$$\text{subject to the constraint}$$
$$\sum_{x \in P_a} (C(F_x)) \;\geq\; \sum_{x \in P_e} (C(F_x))$$

where the expert is confident that P_e is indeed better than P_a. Such a relative constraint would serve two purposes. First, it is much easier to produce. For example, it is much easier to demonstrate a path that prefers a road to a grass field and state 'taking the road is better than taking the field' then it is to state 'taking the road in exactly this manner is better than all other possibilities (including those that also take the road)'. Second, all constraints produced in such a manner are explicitly intended by the expert; the possibility of accidentally implying additional (and possibly inconsistent or suboptimal) constraints is removed.

The problem of learning strict preferences in this manner has been well studied [249], and has been shown to be NP-hard. However, if soft constraints are used, then the problem is tractable. Adding a slack term as before yields

$$\text{minimize} \;\; \mathcal{O}_F[C] = \text{REG}(C) + \zeta \qquad (7.1.3)$$
$$\text{subject to the constraint}$$
$$\sum_{x \in P_a} (C(F_x)) \; - \sum_{x \in P_e} (C(F_x)) \; + \zeta \geq \; 0$$

Moving the constraint into the optimization, and adding a margin yields an optimization problem similar to (5.1.10)

$$\text{minimize} \;\; \mathcal{O}[C] = \; \lambda\text{REG}(C) \; + \sum_{x \in P_e} C(F_x) \; - \; \left[\sum_{x \in P_a} (C(F_x) - L_e(x)) \right] \qquad (7.1.4)$$

This formulation is near identical to the support vector ranking problem [250, 251], except it is not constrained to linear cost functions. As with support vector ranking or an MMP formulation with an optimal constraint, it can be solved through sub-gradient descent. The functional gradient, defined as

$$\nabla \mathcal{O}_F[C] = \lambda \nabla \text{REG}_F[C] + \sum_{x \in P_e} \delta_F(F_x) \;\; - \;\; \sum_{x \in P_a} \delta_F(F_x) \qquad (7.1.5)$$

is identical to (5.1.11) with P_* replaced with P_a. Thus, the implementation of learning from explicit relative examples is identical to the LEARCH implementations from the previous chapters, except instead of accumulating visitation counts along P_*, they are accumulated along P_a. An extra improvement to this formulation would be to use R-LEARCH techniques to allow for noise in either path of the constraint, as well as allowing generalization to similar paths.

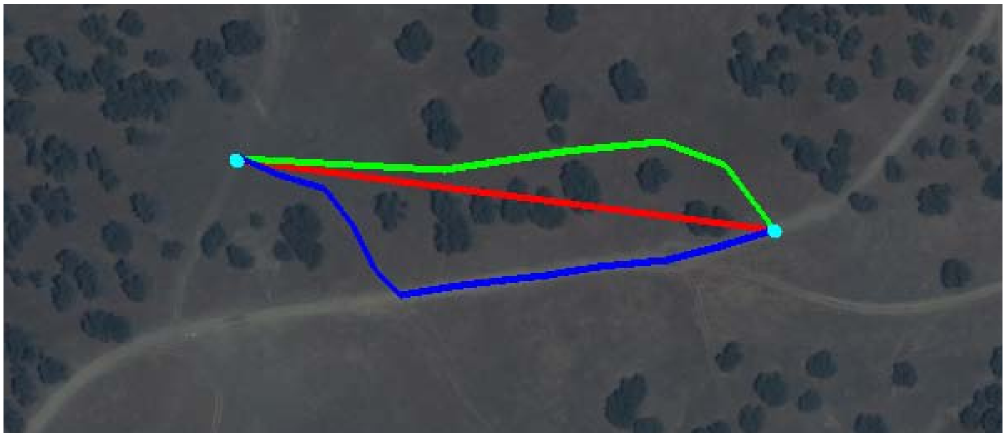


Figure 7.1: 3 Possible paths between two waypoints are shown over overhead imagery. The green path can be used as an example that is relatively better than the red path, and the blue path as one relatively better than the red and green path. Even if the blue path is not the best or most desirable path, the relative constraints that it implies can be useful for learning.

Therefore, one alternative approach to collecting expert demonstration is to request two trajectories from an expert, P_e and P_a, such that one is clearly better than the other. Alternatively, rather than the expert producing P_e and P_a, he could be provided with paths P_1 and P_2 between two waypoints, and asked to identify which one is preferable. Such a procedure could reduce the burden on an expert, as well as the possibility of error in demonstration. A similar approach was widely used in the DARPA Learning Locomotion program in order to learn a cost function over individual footstep placements for the Little Dog robot. The general approach was to present the trainers with multiple terrain patches, and to record their preferred terrain patch with respect Little Dog. This approach was adopted by several competing teams, demonstrating its ease in facilitating cost function learning [252, 253, 254].

7.1.2 Acceptable Examples

In scenarios that are difficult for a human expert to understand or interpret, relative constraints can be an effective way to provide feedback without the necessity of identifying the single most desirable solution. However, in certain instances, it may be too difficult to make even these determinations. Systems that operate in high dimensions are especially difficult. For example, an expert may be able to demonstrate a successful trajectory for a high degree of freedom robotic arm, or for a vehicle traveling at high speeds (where dynamics are in play). However, the expert may not be able to say with confidence whether the demonstration was better than any other approach that was also successful, let alone that it was the single most desirable trajectory.

In such difficult cases, there is still potential to use relative constraints to learn from expert feedback. The key is the definition of successful. Imagine a high speed robotic vehicle traveling between waypoints. If an expert demonstrates a trajectory that moves the vehicle from a specified start to a goal, then such a trajectory is clearly acceptable. That is, it is something the expert would be willing to allow the vehicle to do (otherwise, it would not have been demonstrated). In contrast, a trajectory that does not succeed in getting the vehicle to the specified goal, or traverses terrain that it should not, is almost by definition unacceptable.

Therefore, for any start and end condition two classes of examples can be defined. Acceptable, cromulent or satisfactory examples are ones which an expert would allow the robot to perform. Unacceptable or unsuitable examples are ones which the expert would never allow or want the robot to perform. While this distinction provides no information about the relative desirability of examples within a class, there is a clear distinction between classes: no unacceptable example should ever be preferred to an acceptable one.

Formally, if ${\mathbb{P}_{\mathbb{A}}}^{s,g}$ is defined as the set of acceptable paths between s and g, and ${\mathbb{P}_{\mathbb{U}}}^{s,g}$ as the set of unacceptable paths, then the resulting constrained optimization over C is

$$\begin{gathered}\text{minimize } \mathcal{O}_F[C] = \textsc{Reg}(C) \\ \text{subject to the constraints} \\ \sum_{x \in P_U} (C(F_x)) \; > \; \sum_{x \in P_A} (C(F_x)) \\ \forall P_U \in {\mathbb{P}_{\mathbb{U}}}^{s,g}, \; P_A \in {\mathbb{P}_{\mathbb{A}}}^{s,g}\end{gathered} \tag{7.1.6}$$

Adding a margin and rearranging the constraint yields

$$\begin{gathered}\text{minimize } \mathcal{O}_F[C] = \textsc{Reg}(C) \\ \text{subject to the constraints} \\ \sum_{x \in P_U} (C(F_x)) - \sum_{x \in P_A} (C(F_x)) > \sum_{x \in P_U} L_e(x) \\ \forall P_U \in {\mathbb{P}_{\mathbb{U}}}^{s,g}, \; P_A \in {\mathbb{P}_{\mathbb{A}}}^{s,g}\end{gathered} \tag{7.1.7}$$

With the exception of the loss scaled margin, this optimization is identical to the standard support vector machine formulation. This implies that the separation into acceptable and unacceptable groups can indeed be viewed as a classification problem.

From an implementation standpoint, the functional gradient (and therefore the necessary visitation counts to collect) are unchanged from relative constraints. The total number of constraints is quadratic in the number of examples. However, it is not necessary to enforce all of these constraints. Instead, it need only be enforced for P_U* (the lowest cost unacceptable path), resulting in a linear number of constraints. Technically speaking, a single constraint would suffice, using only the highest cost path in ${\mathbb{P}_{\mathbb{A}}}^{s,g}$ at any time; however given that every example path (in either class) provides a noisily labeled set of individual examples, there is potentially benefit to generalization in using more constraints.

7.1.3 Lethal Examples

A key distinction between inverse optimal control based approaches to learning preference models and many other approaches is the notion that there is not necessarily a single correct cost value; rather only correct behavior produced by a set of costs. However, there is one specific case where often the correct cost can be identified. This is in the case of true obstacles: terrain features that a robot should never attempt to traverse. Despite the prevalence of continuous terrain costs, many planning systems still have a notion of an impassable or lethal cost, such that transitions through

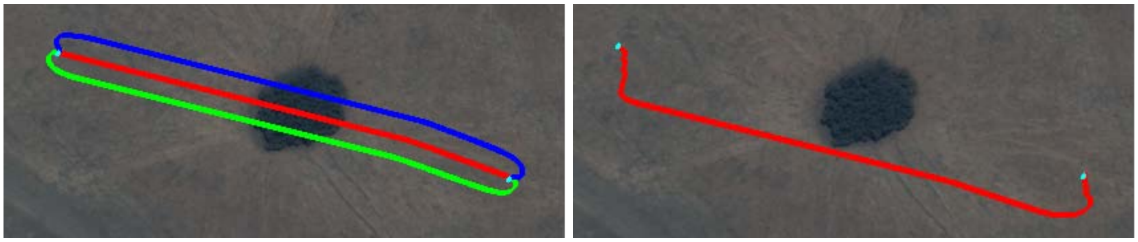


Figure 7.2: 3 unacceptable examples (left) shown along with 1 acceptable example (right). This set of examples implies 3 relative constraints (between each unacceptable example and the acceptable example); alternatively this can be viewed as a classification problem between unacceptable and acceptable paths.

lethal states can not even be considered by a planner.

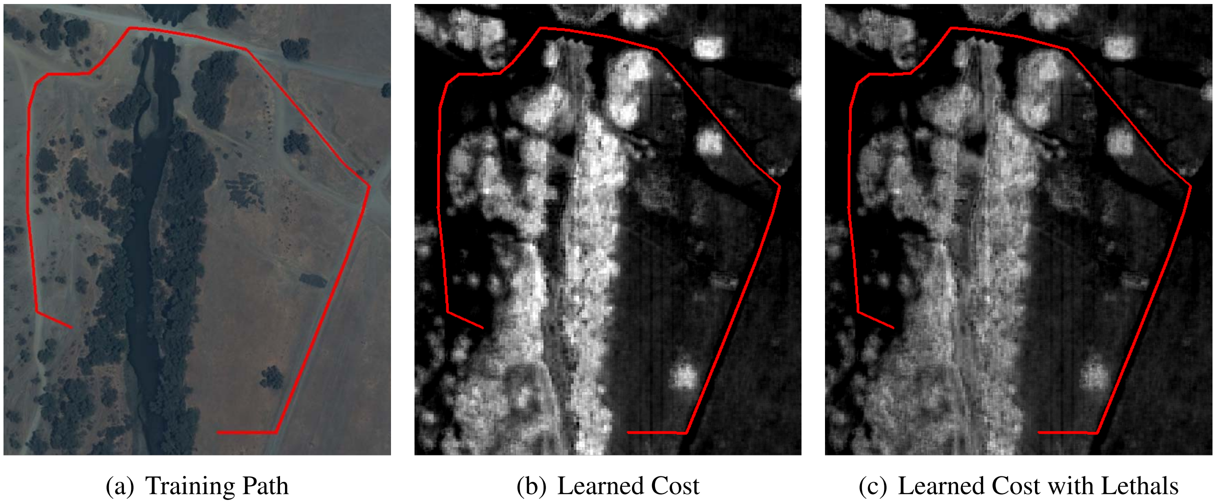
However, the power to consider a state (or action) as lethal comes with drawbacks. Specifically, any errors in lethal identification (whether a false positive or negative) can drastically affect behavior. For example, the Crusher system originally made use of lethal perception costs to help ensure safety. However, lethal false positives would often close off narrow but traversable paths, forcing the robot to explore further (and sometimes take a more dangerous route). Also, depending on lethals to ensure safety became something of a crutch, and lethal false negatives would often lead to safety interventions. For these reasons, explicit lethal costs were removed from the Crusher system in later iterations[1].

However, lethals can still be useful, if they are accurately identified. Not only can they help to ensure a robot's safety, they can also help to account for a limited dynamic range in a cost function. Cost functions learned from demonstration are especially prone to dynamic range issues (Figure 7.3). The problem is that even if a particular obstacle should be avoided at all costs, any (feasibly demonstrable) example path will only imply a lower bound. Driving 100m over a grass to avoid a 1m square rock only implies a (relative) cost of one hundred; if the desired ratio is closer to one thousand or one million, it is unlikely that any feasible example will actually exist in the environment that would imply such a bound.

In order to solve the problem of lethal identification, one approach would be to explicitly label such examples. That is, an expert would label patches of terrain as lethal or non-lethal, and a simple binary classifier could then be used as an augmentation to the existing cost function. However, unless such a classifier were completely accurate, it would still suffer problems with the unintended consequences of errors. The idea of explicit labeling of lethals can still be useful however; the key is in how such labels are used. If they are integrated in a way that also takes into account existing expert demonstration, then the fusion of all training data is likely to reduce the consequences of errors. That is, if a specific patch of terrain is labeled as lethal, and no training examples ever traverse similar terrain, then the cost function could consider it lethal. If however, a terrain patch labeled as lethal is indistinguishable (to the chosen family of cost functions) from terrain that an expert traverses in a demonstration, then these inconsistent training inputs can be balanced (that is, the cost raised as much as possible on that terrain, while still allowing the example to be reproduced).

Assume that for a given planner, there is a maximum cost value C_{max}, at or above which all

[1] Chapter 3 also described how lethals created problems in Urban Challenge vehicles



(a) Training Path (b) Learned Cost (c) Learned Cost with Lethals

Figure 7.3: An case where an example path does not imply significant dynamic range of dangerous terrain. Although the learned cost function reproduces the demonstrated behavior, it does so in a manner that does not generalize well (that is, cost on the water is too low and cost on the trees too high). Labeling just a few pixels of water as lethal results in higher cost on water, and lower cost on trees.

costs are treated as lethal. For many planners, including Field D*, this is how lethal costs are actually implemented. For planners without explicit lethals, C_{max} is then the closest a cost value could ever come to a true lethal. If a labeled set of lethal examples $\mathbb{L}$ are provided by an expert, than a reasonable way to seek a cost function would be to do a normal MMP optimization, while also enforcing the lethality of all such labeled examples

$$\begin{gathered}\text{minimize } \mathcal{O}_F[C] = \textsc{Reg}(C) + \text{MMP}(C) \\ \text{subject to the constraints} \\ C(x) \geq C_{max} \quad \forall x \in \mathbb{L}\end{gathered} \tag{7.1.8}$$

MMP represents the contribution to the objective of the standard MMP constraints (condensed for the sake of clarity). However, there is a fallacy with this formulation. If a lethal example and a standard MMP example are in conflict (that is, terrain patch is labeled as lethal, and yet a training example drives through similar terrain), that terrain would be considered lethal (match the hard lethality constraints). However, the fact that an expert chose to drive a vehicle through such terrain would seem to indicate that that is not the case. To allow the conflicting examples to be balanced in such cases, the constraints can be softened through a slack penalty

$$\begin{gathered}\text{minimize } \mathcal{O}_F[C] = \textsc{Reg}(C) + \text{MMP}(C) + \sum_{x \in \mathbb{L}} \zeta_x \\ \text{subject to the constraints} \\ C(x) + \zeta_x \geq C_{max} \quad \forall x \in \mathbb{L}\end{gathered} \tag{7.1.9}$$

Moving the lethality constraint into the objective yields

$$\text{minimize } \mathcal{O}_F[C] = \textsc{Reg}(C) + \text{MMP}(C) + \sum_{x \in \mathbb{L}} (C_{max} - C(x)) \tag{7.1.10}$$

with functional gradient

$$\nabla \mathcal{O}_F[C] = \lambda \nabla \textsc{Reg}_F[C] + \nabla \text{MMP}_F[C] - \sum_{x \in \mathbb{L}} \delta_F(F_x) \tag{7.1.11}$$

This gradient implies a very simple addition to the LEARCH algorithm that can allow lethal examples: states labeled as lethals simply have their (positive) visitation counts incremented at each iteration, if they are not already sufficiently high cost. During the projection of the functional gradient (that is, each regression step to learn a modification to the cost function) these extra visitation counts will result in an increase of the cost at the labeled states, as long as there are not in conflict with other examples. The result is a simple algorithm that can help to account for dynamic range problems, and potentially further simplify training data collection.

7.1.4 Expert Feedback without Demonstration

The addition of relative, acceptable and lethal examples to standard (optimal) examples has the potential to ease the burden on an expert demonstrator by lowering the threshold on the quality of expert input for it to be useful. However, (except for lethal examples) an expert must still produce the actual demonstration. In the case of relative or acceptable examples, this implies producing not only the preferred or acceptable example, but the example to be avoided. When demonstration requires operation of an actual robot, it may not be possible or desirable to actual generate such examples.

However, the combination of these new forms of expert feedback offers an intriguing possibility: that of providing expert feedback to improve the system without the expert ever actually demonstrating any behavior to the robot. That is, both positive and negative example behaviors could be produced automatically, and then simply rated by the expert. Such an approach would be especially useful for robotic systems that are too complicated for an expert to operate, or when such demonstration is simply too arduous.

In the case of a mobile robot, such a training approach could proceed as follows. The robot (through some combination of normal optimal examples and lethal examples) is given a small training set sufficient for it to operate autonomously in a safe manner with close expert observation. Then, the robot is taken to the field for testing. During testing, any time the robot attempts a behavior that the expert deems unacceptable, the robot is stopped and the robot's plan recorded. The robot then raises the cost specifically of the unacceptable plan; this could take the form of simply raising the costs on the exact states (not their feature vectors) that the plan would have traversed. This process repeats until an acceptable plan is reached. The result is a set of acceptable behaviors, and one or more corresponding unacceptable ones (per acceptable example). Such a procedure could also be used to produce multiple examples that are acceptable, but for which the expert can express a preference between.

Such a set of procedures could prove useful even when explicit demonstration of the robot is possible. At times, it may prove easier to simply correct the robot's behavior than to fully demonstrate new behavior. Correcting errors has the advantage that the training data is by definition useful, since it specifically addresses cases the robot does not yet properly understand. In addition, it could be a far less time consuming way of gathering expert feedback. This is especially important during robotic field testing that consumes a large amount of resources, human or otherwise.

This overall procedure is potentially quite similar to the approach of Reinforcement Planning [255], in which trajectories and the reward they produce from a (known) reward function are used to improve a cost function. In the case of a mobile robot, unit positive and negative reward can be used to signify achieving a waypoint or the robot colliding with an obstacle respectively. However, this of course requires the robot to actually make mistakes. In practice, an expert stopping a robot before it made a mistake would almost certainly be used, and such an intervention could be tied to a negative reward. However, as long as an expert needs to be in the loop (to ensure vehicle safety), the expert might as well provide information indicating specifically what went wrong, and how to correct it. This same argument applied to regular reinforcement learning (Section 4.3). Essentially, a constraint based approach is likely to be more efficient than reinforcement planning, as blame assignment can potentially be made explicit through the use of lethal examples (that is, an expert can not only stop the robot, but specify exactly what terrain feature or driving maneuver caused the intervention), as well as the desired corrections explicitly demonstrated.

7.2 Active Learning for Example Selection

Previous experiments have depended on the expert to provide not only the actual demonstrations used to teach the system, but also to choose what to demonstrate. This puts additional burden on the expert, and raises the bar for who can qualify to train the system; the expert must now have sufficient knowledge of the actual system internals to know which examples are most useful. In addition, the expert must understand the structure of the example to fully comprehend how useful it is or not. For example, driving a robot around a rock in an open field is useful, but avoiding the same rock in a field of tall grass is even more useful (as the chaining of constraints helps increase dynamic range).

Having the expert choose what to demonstrate can also affect the quality of the final training set. It has been observed empirically that a good deal of expert demonstration proves to be redundant (see Figures 5.10, 5.16 and 7.5). That is, the same concepts and constraints are repeatedly demonstrated. While this does help to potentially fine tune such concepts, it also is an inefficient use of an expert's time, especially when the possibility remains that there may still be useful and important concepts that have not been properly demonstrated. If the amount of expert interaction is held constant, such inefficient use of time will lower the overall quality of the training set. Additionally, forcing the expert to decide what examples to provide results in a distribution of training data that may be significantly different than the distribution the robot will encounter during actual operation. While it is important to capture an expert's intuition as to which examples are important, this can be taken to the extreme if an expert repeatedly demonstrates an important (but rare) preference, potentially skewing behavior in more common cases.

These issues are not unique to learning from demonstration, but are common to all forms of supervised learning. As a result, the field of *Active Learning*, where the learner is at least partially responsible for choosing which training data to obtain, has received focus in order to aid in the efficient selection of a good training set. Learning preference models from demonstration could potentially be aided by the incorporation of active learning techniques, that requested demonstration in specific scenarios (e.g. how to drive in a certain type of terrain, or how to perform a specific driving maneuver). Such techniques could reduce the cognitive burden on an expert, as well as increasing the pool of potential experts. In addition, active learning could help to ensure that each additional example was maximally useful and not redundant.

Active learning has been previously investigated with respect to structured prediction problems. Much of this work has focused on the context of sequence labeling or semantic parsing tasks [256, 257]. One of the ideas to grow out of this work is to consider the cost associated with obtaining a labeling [258, 259, 260]. For a structured prediction problem, obtaining labelings in certain instances may be far more difficult than in others; if the goal is to ease the burden on the expert, this cost should also be taken into account. Additionally, it is possible that the current hypothesis may be partially correct, and that this may aid the expert in producing a labeling [258]. These ideas are all very relevant to active learning in the learning from demonstration domain.

Active learning with respect to certain forms of learning from demonstration has received some previous attention. The approaches of [261, 262] each allow a robot to request instruction whenever it is not sufficiently confident in a learned action it is about to execute, based on its similarity to previously observed actions[2]. These approaches represent a form of query filtering: the learner encounters tasks one at a time, and must decide whether to request expert aid for each task. This requires the expert demonstrator to be 'on call' to provide demonstration. This mode of operation could potentially be quite useful, especially once an initial training set has been produced. However, it would prove quite time consuming for the expert if used exclusively. Alternatively, pool based approaches (that request expert instruction over a small subset of a large, unlabeled data set) would minimize the initial expert interaction necessary to train a robust system. The techniques presented in this chapter are primarily pool based approaches; however they could also be applied for query filtering in an online setting when an expert is available.

7.2.1 Active Learning through Novelty Reduction

In an ideal setting, a set of training demonstration provided by an expert would come from the same distribution of examples as that which the robot would encounter while operating autonomously. This is quite unrealistic, especially since it is rarely the case that the training and operating environments are identical. A more realistic desire would be that the two distributions at least cover the same areas of the example space: that is, that for every possible scenario the robot could find itself in, it has been trained on something sufficiently similar such that it can properly generalize. Unfortunately, this is also difficult ensure, as the example space is hard to quantify. Instead of the example space, the feature space over states and actions can be used as a proxy. Therefore, a potential goal of active learning could be to ensure that for any type of terrain or driving scenario the robot may encounter, it has at least seen something similar in the training set.

In order to understand what the robot has not been exposed to, it is necessary to model what the robot has been exposed to. This problem can be phrased as a form of density estimation over the feature space of states and/or actions. Then, examples can be chosen from regions with low density. Such regions are often referred to as anomalous or novel, in that feature vectors that have not been frequently seen during training represent something novel to the learner. Novelty detection in and

[2] In this regard these approaches are not considered generalizable learning from demonstration

of itself is a rich field of research [263, 264], and has seen many applications to mobile robotics [265, 266, 267, 268, 269]. Dima et al. [270, 271] specifically propose density estimation in the context of active learning for robotic perception systems. Specifically, a large dataset consisting of a robot's recorded perceptual history is analyzed to identify the most unlikely single percepts (given the entire history). These percepts are then provided to human experts to be labeled for use in training a terrain classification system. This process is then repeated to produce a full training set that properly covers the entire space of percepts.

This same general approach could also be applied to learning from demonstration. In this context, all terrain or action features observed during previous training would be analyzed. Then, the entire (unlabeled) set of features would be analyzed to detect novel cases. Expert demonstration could then be requested for novel structures. Such a novelty based active learning procedure would proceed in 3 phases

1. **Initial Training:** Perform learning from demonstration on an existing training set, and build a data set of every feature vector encountered during learning.

2. **Density Estimation:** From the data set, build a density model, such that for any test feature vector, a measure of how similar it is to already seen data can be produced.

3. **Example Selection:** Once a density model has been computed, example problems are selected such that the resulting demonstration is likely to provide useful information about novel areas.

This process would then be iterated to continually add the most novel examples, and is shown in more detail in Algorithm 9.

An additional step that could be inserted to this process is that of dimensionality reduction before density estimation. Performing density estimation in a high dimensional space (such as a perceptual feature space) is computationally expensive, as well as prone to potential overfitting. Dimensionality reduction can alleviate both of these problems, and is quite common as a precursor to density estimation. Dimensionality reduction is itself another vast area of research [272], with Principal Components Analysis (PCA) being perhaps the single most common approach.

As for performing density estimation in the (reduced) feature space, there are two main classes of approaches: parametric and non-parametric. While [270] uses non-parametric density estimation, parametric approaches are generally much faster at test time, allowing for easier application in an online setting (to allow for query filtering as well as a pool approach). Therefore, parametric density estimation is used in this context. In the end there are many different specific algorithms that could be applied for both dimensionality reduction and density estimation, each with their own advantages and tradeoffs. In the end, any approach could be applied that results in the ability to produce some sort of novelty measure for various test points.

Example selection is the most unique part of this procedure. In the context of learning terrain preferences, this would consist of producing waypoints such that the resulting example path is likely provide new information about novel terrain. In the context of learning driving styles or action preferences, this would consist of waypoints such that the resulting example trajectory is likely to involve a novel maneuver. The challenge is then to try and anticipate the demonstration that is likely to result from a specific test problem. The key insight to solving this problem is that there already exists a seed set of expert demonstration, and a cost function learned from the set that

Algorithm 9: Novelty Based Active Learning

```
Inputs: Existing Set of Examples $\mathcal{P}$, Entire Feature History $\mathcal{F}_H$, Prior Cost Function $C_0$
$[C, \mathcal{D}]$ = LEARCH($\mathcal{P}, \mathcal{F}_H$) ;   // $\mathcal{D}$ is subset of $\mathcal{F}_H$ encountered learning $\mathcal{C}$
$R$ = learnDimReductionFunction($\mathcal{D}$);
$D_{pdf}$ = learnDensityEstimationFunction($R(\mathcal{D})$);
for $i = 1...K$ do
    $W_i$ = generateRandomWaypoints();
    $N_i$ = evaluateNoveltyOfPath($D_{pdf}, R(\mathcal{F}_H)$,planPath($\mathcal{F}_H, C, W_i$));
    $N_i = N_i +$ evaluateNoveltyOfPath($D_{pdf}, R(\mathcal{F}_H)$,planPath($\mathcal{F}_H, C_0, W_i$));
$\mathcal{W}$ = chooseActiveLearningWaypoints($W_1, N_1, W_2, N_2, ..., W_K, N_K$);
return $\mathcal{W}$
```

seeks to reproduce the example behavior. Since the cost function has been learned to try and imitate the expert, it can also be used to try and predict their future behavior [240]. Therefore, between any two test waypoints the planned trajectory under the current cost function is a reasonable prediction for the experts future demonstration. Therefore, the aggregate novelty along the current planned path can be used as a measure of the overall novelty of the example problem. Regardless of the expert's actual demonstration, something will be learned about this novel example: if the expert does produce the predicted example, then an example of preferring previously novel terrain or actions has been added to the training set, and if the expert produces a different behavior then an example of *not* preferring novel terrain has been added.

Therefore, the following example selection heuristic is proposed. First, select a large set of random waypoint pairs, and plan behaviors under the current cost hypothesis. Next, evaluate the aggregate novelty along each plan. Finally, choose a random subset of the waypoint pairs to present to the expert, with the likelihood of a pair's selection related to its aggregate novelty. As the resulting demonstrations are likely to provide information about previously novel regions of the feature space, they will help to ensure at least minimum coverage of said space. An additional option for this heuristic is to evaluate aggregate novelty under two possible plans: that under the current cost hypothesis, and under the initial cost hypothesis (C_0). High cost regions of the feature space are often novel simply because an expert never demonstrates behavior that contains them; also considering the initial cost hypothesis can produce waypoint pairs that continually force the expert to demonstrate their aversion to such areas, thus lowering their novelty.

In addition to providing a useful metric for aiding in offline active learning, a novelty function learned through the above procedure would also be useful in an online active learning. Such a novelty function would allow for the identification of terrain features or maneuvers on which the robot has not been sufficiently trained. As demonstrated in [267, 269], such novelty functions can be useful in the autonomous operation of a mobile robot by preventing it from encountering scenarios for which it is unprepared. Additionally, it could also be used in a query filtering approach if an expert is available. That is, if the robot intended to drive through a low cost but novel section of terrain, it could first request expert 'permission' to do so before proceeding (and then learn from the resulting answer).

Algorithm 10: Uncertainty Based Active Learning

```
Inputs: Existing Set of Examples $\mathcal{P}$, Entire Feature History $\mathcal{F}_H$, Prior Cost Function $C_0$
$C$ = LEARCH($\mathcal{P}, \mathcal{F}_H$);
for $j = 1...N$ do
    $\hat{\mathcal{P}}_j$ = chooseRandomSubsetWithReplacement($\mathcal{P}$);
    $\hat{C}_j$ =LEARCH($\hat{\mathcal{P}}_j, \mathcal{F}_H$);
for $i = 1...K$ do
    $W_i$ = generateRandomWaypoints();
    $U_i$ = evalUncertaintyOfPath($\hat{C}_1, ..., \hat{C}_N, \mathcal{F}_H$,planPath($\mathcal{F}_H, C, W_i$));
    $U_i = U_i +$ evalUncertaintyOfPath($\hat{C}_1, ..., \hat{C}_N, \mathcal{F}_H$,planPath($\mathcal{F}_H, C_0, W_i$));
$\mathcal{W}$ = chooseActiveLearningWaypoints($W_1, U_1, W_2, U_2, ..., W_K, U_K$);
return $\mathcal{W}$
```

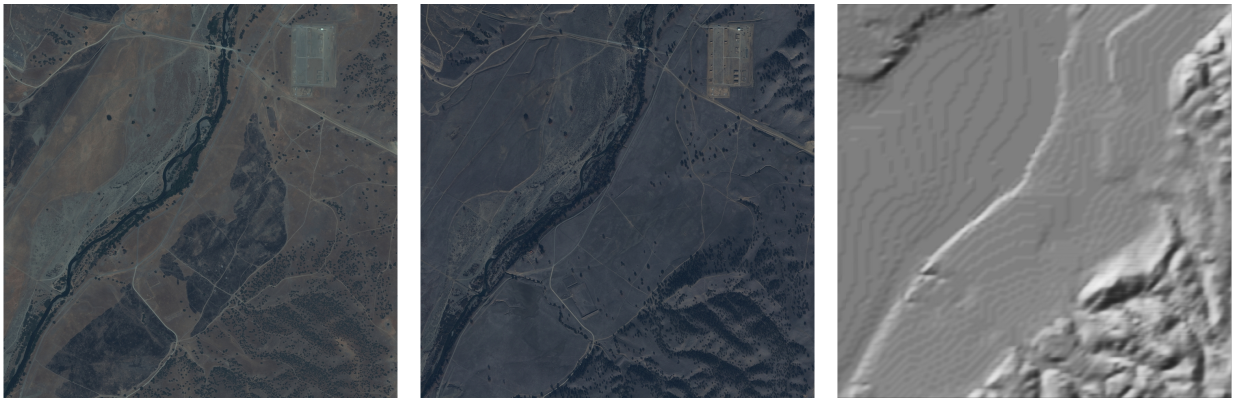
7.2.2 Active Learning through Uncertainty Reduction

As opposed to specifically seeking expert feedback on novel examples, another approach is to request expert information on examples for which the learner already has information, but remains uncertain. There is of course a relationship between novelty and uncertainty, in that a learner is likely (but not guaranteed) to be uncertain about truly novel examples. However, novelty is not the only source of uncertainty; it can also come from similar examples with conflicting labels or demonstrated behavior.

When a learner explicitly models its own uncertainty, an active learning technique known as *Uncertainty Sampling* [273] can be applied. However, explicit uncertainty sampling in the context of learning preference models would limit the choice of regressor to Gaussian Processes or Bayesian linear regression. Gaussian Processes are not a good choice for learning a cost function when real time cost production is required, as they are quite expensive to apply when trained on large datasets. Bayesian linear regression is parametric; however this in turn limits the hypothesis space of cost functions. Therefore, in order to retain the agnosticism to the specific form of cost function that LEARCH provides, explicit uncertainty sampling is not an option.

However, there is another approach that can work with general cost functions. The *Query by Bagging* approach [271] combines the idea of *Query by Committee* [274, 275] with the idea of *Bagging* [276] to measure the uncertainty still inherent in a training set for a particular class of learner. The idea is to train multiple learners on different random subsets (with replacement) of the available training set, and see where they disagree. In the context of learning preference models, the analog would be to learn multiple preference models from different subsets of the available demonstration. Then, the uncertainty of a particular example would be approximated as the variance of the different cost functions over said example.

Therefore, an uncertainty based active learning approach could be implemented that is nearly identical to the novelty based approach. First, an uncertainty model would be constructed based on existing expert demonstration. This model would then be used to evaluate plans (under both the current hypothesis and C_0) between random waypoints. Waypoint pairs where the demonstration is predicted to traverse high uncertainty areas would then be presented to the expert as a request for explicity demonstration, and the process repeated. Such a procedure is shown in Algorithm 10. Also as before, a final uncertainty model could be used in an online setting for query filtering.



(a) Satellite Imagery Summer 2004 (b) Satellite Imagery Fall 2007 (c) Shaded Relief Elevation Map

Figure 7.4: The 9 km^2 test site used for these experiments

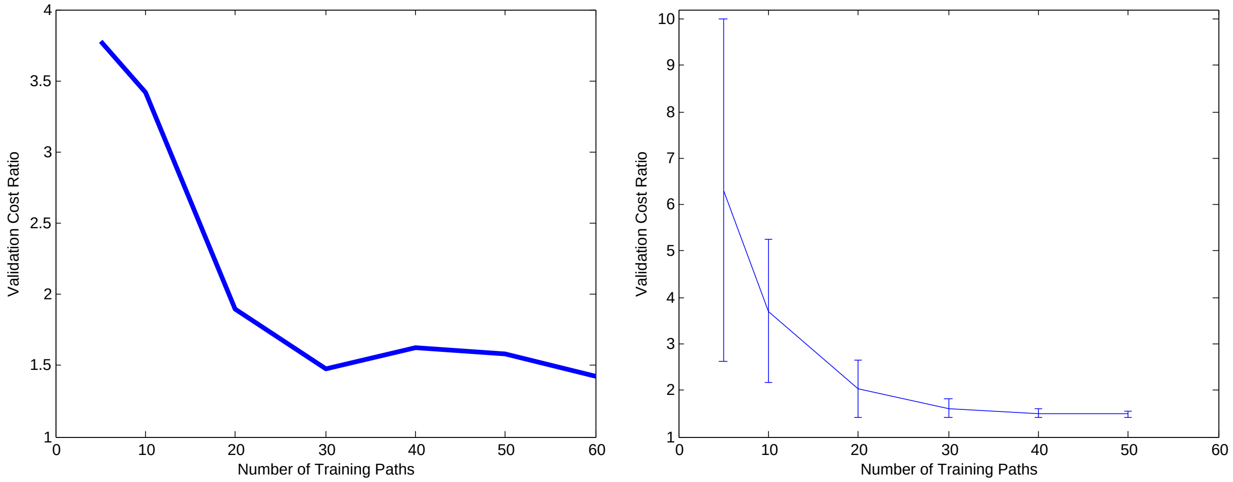
As explored in [277] there are yet more additional uses for knowledge of uncertainty in a cost function.

Alternatively, another approach to example selection is possible. Unlike with novelty based active learning, the uncertainty model provides a metric in the same units as cost. Therefore, the uncertainty model can be used to estimate upper and lower bounds on the 'true' cost for any test point (the actual observed bounds could also be used). Rather than choosing waypoints that are likely to involve demonstration through regions of high uncertainty, waypoints could be chosen such that the predicted plan under the upper bound of the cost hypothesis is significantly different than the predicted plan under the lower bound hypothesis. This difference can easily be expressed as the cost ratio under the current (mean) cost hypothesis.

Practically speaking, this method of example selection would have two advantages. First, it would be likely to choose examples such that the cost uncertainty actually affects the plan; otherwise, waypoints that involve traveling through uniform regions of high uncertainty could be favored. Second, instead of simply producing a pair of waypoints and then required explicit expert demonstration, this approach produces two possible plans between a pair of waypoints; the expert then need only provide a relative preference as in Section 7.1.1.

7.3 Experimental Results

All experiments in this chapter were performed on overhead data for a large (9 km^2) test site. Although this results in a static learning from demonstration task, all extensions to the dynamic problems are straightforward. The test site consists of several distinct terrain features such as short and tall grass, bushes, sand, water, different types of trees, and rolling hills. There are also man made features such as roads, buildings and areas used for controlled burns. The overhead data consists of two sets of satellite imagery and a digital elevation map (Figure 7.4). Candidate cost functions are evaluated using the resulting cost ratio on a large validation set of 250 example paths.



(a) Original Order (b) Random Subsets

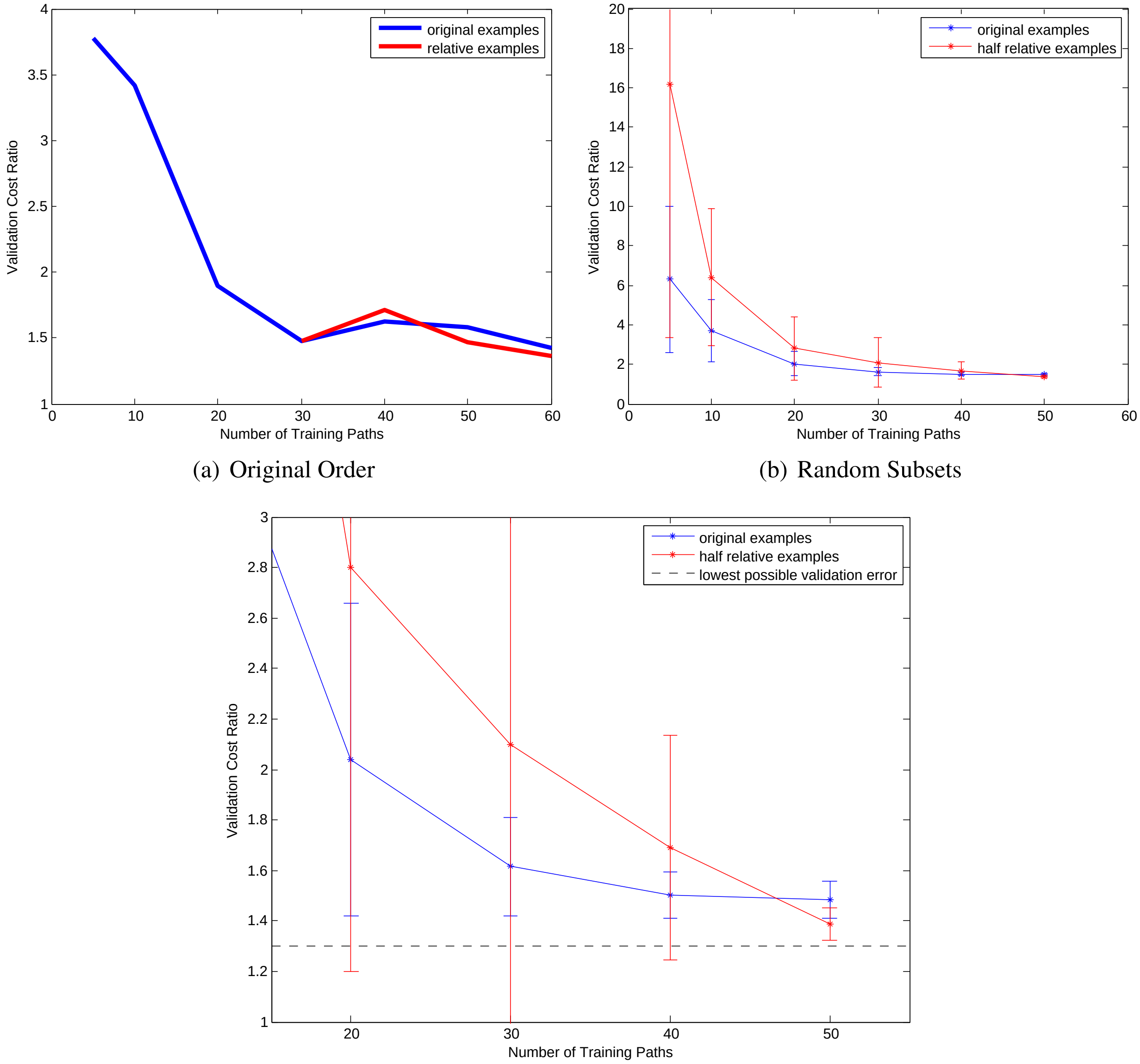
Figure 7.5: Validation performance for increasing numbers of example paths, in (a) their original order and (b) random permutations (error bars indicate one standard deviation). Over time, there are clear diminishing returns to adding additional (expert chosen) examples.

7.3.1 Different Forms of Expert Demonstration

In order to first quantify the redundancy and inconsistency in expert demonstration, the following experiment was performed. First, 60 example training paths were demonstrated by an expert. For different numbers of example paths (in their original order), a cost function was learned using R-LEARCH and applied to the entire test site. Next, random subsets of 50 training paths were chosen, and were used to train cost functions using different numbers of examples in a random (permuted) order. Figure 7.5(a) shows the ordered version of this experiment. For the first half of the training set, adding more examples significantly improves validation performance, demonstrating that new and useful preferences are being demonstrated for the first time, or usefully reinforced. However, the second half of the training set results in only minimal improvement, and over certain intervals actually decreases performance. This is evidence that the second half of the training set was redundant and/or inconsistent with the first half.

Figure 7.5(b) shows the same experiment with random ordering of training examples, along with error bars of one standard deviation. The overall trend of diminishing returns is repeated, indicating that the redundancy in the ordered experiment was not explicitly due to ordering itself. Regardless of what order the training examples are provided in, after about 2/3 of the total training set performance does not significantly improve. Also of note is the incredibly high variance with small numbers of training examples. This is further proof that when examples are simply chosen by an expert, the only way to ensure consistency and stability is to sacrifice efficiency (and provide a large training set).

Next, an experiment was performed to determine the effect of mixing relative examples (Section 7.1.1) with regular examples. For this experiment, the first half of the original training set was left alone, and used to train a single cost function. This cost function was then used to plan paths between every waypoint pair used in the second half of the training set. These pairs of paths then became relative constraints. That is each expert drawn path, rather than being a regular (opti-



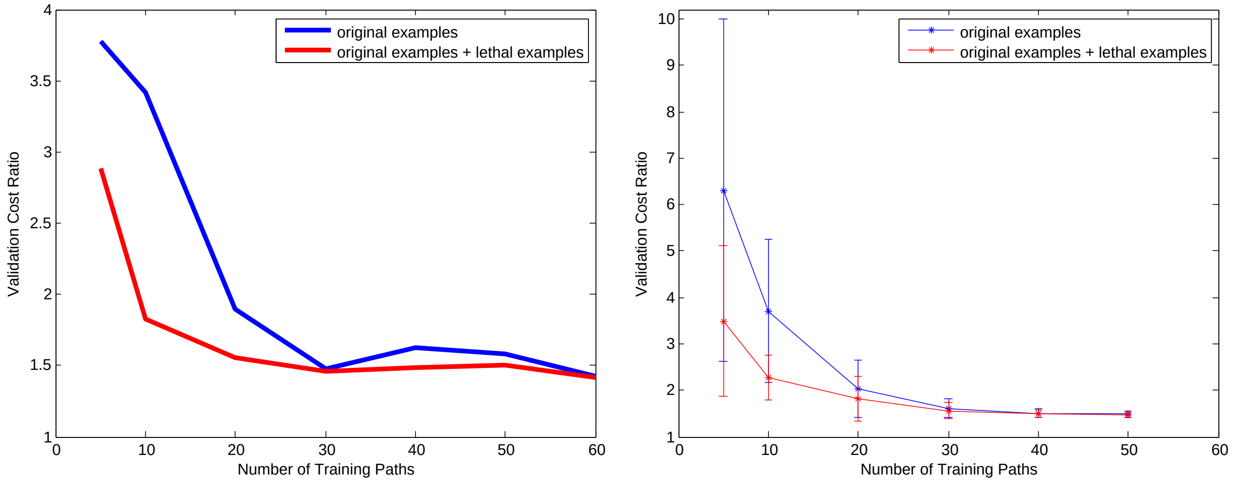
(a) Original Order

(b) Random Subsets

(c) Random Subsets (Zoomed)

Figure 7.6: Validation performance when replacing the second half of the training set with relative examples. When adding to the training set in its original order (a) performance is not significantly altered, indicating relative constraints may be a more efficient way of achieving the same results. When random orderings (b) are applied, performance of small training sets is significantly worse; however, once the training set is of sufficient size the performances are first equivalent, and then better when using relative constraints, shown in (c) to be much closer to the theoretical minimum validation error.

mal) example, formed a relative example with the planned path under the cost hypothesis (with the constraints implying the expert drawn paths must be considered lower cost). Therefore, this new training set contained the exact same expert demonstration as before; however, the second half of the training set was now interpreted differently. Figure 7.6(a) shows the results of this experiment,



(a) Original Order

(b) Random Subsets

Figure 7.7: Validation performance with the addition of lethal training examples. The inclusion of lethal examples both improves average performance while decreasing variability when using small training sets.

with examples added in their original order. The performance is nearly identical to the original experiment (using all paths as optimal examples). This indicates that the preferences implied from the second half of the training set could just as easily be learned if the examples were relative as opposed to optimal. Such a result could lead to increased efficiency when producing a training set. That is, once a point of diminishing returns is reached, an expert does not need to provide the *most* desirable path, simply a path that is more desirable than the current hypothesis.

Figure 7.6(b) shows the results of performing the same experiment with random permutations of the training set. The examples that were converted to relative examples could appear at any point in the permutation. The validation performance as a function of the size of the training set at first performs as one might expect. That is, lessening the constraint implied by some of the examples results in worse initial performance, as well as more variability in that performance (for every data point except the last, the increased variance is statistically significant at the 1% level). This is intuitive, as there is less information in the training set. As more examples are added (and more of them are likely to be optimal examples) the performance catches up to the original experiment (with the differences being statistically insignificant at 20,30 and 40 examples). However, something interesting happens once the training set grows sufficiently large (Figure 7.6(c)). At 50 examples, the performance using some relative examples is better on average than using all optimal examples. This difference is significant in both meanings of the word: not only is it significant at the 1% level, but the decrease in cost ratio (from 1.48 to 1.39) brings the final performance significantly closer to the theoretically lowest possible error (the error when training directly on the validation set) of 1.30. This difference implies that not only could relative constraints be a more efficient way of adding new information to an sufficiently large existing training set, they may be a more stable way. This stability comes from reduced inconsistency; by implying fewer (but clearer) preferences, the chances of accidentally implying inconsistent preferences are lowered.

A similar set of experiments was also performed to observe the effect of adding lethal examples

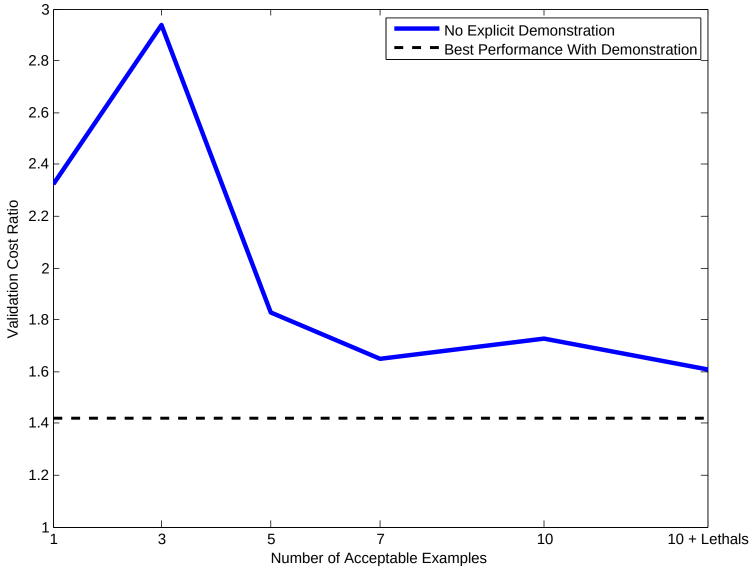
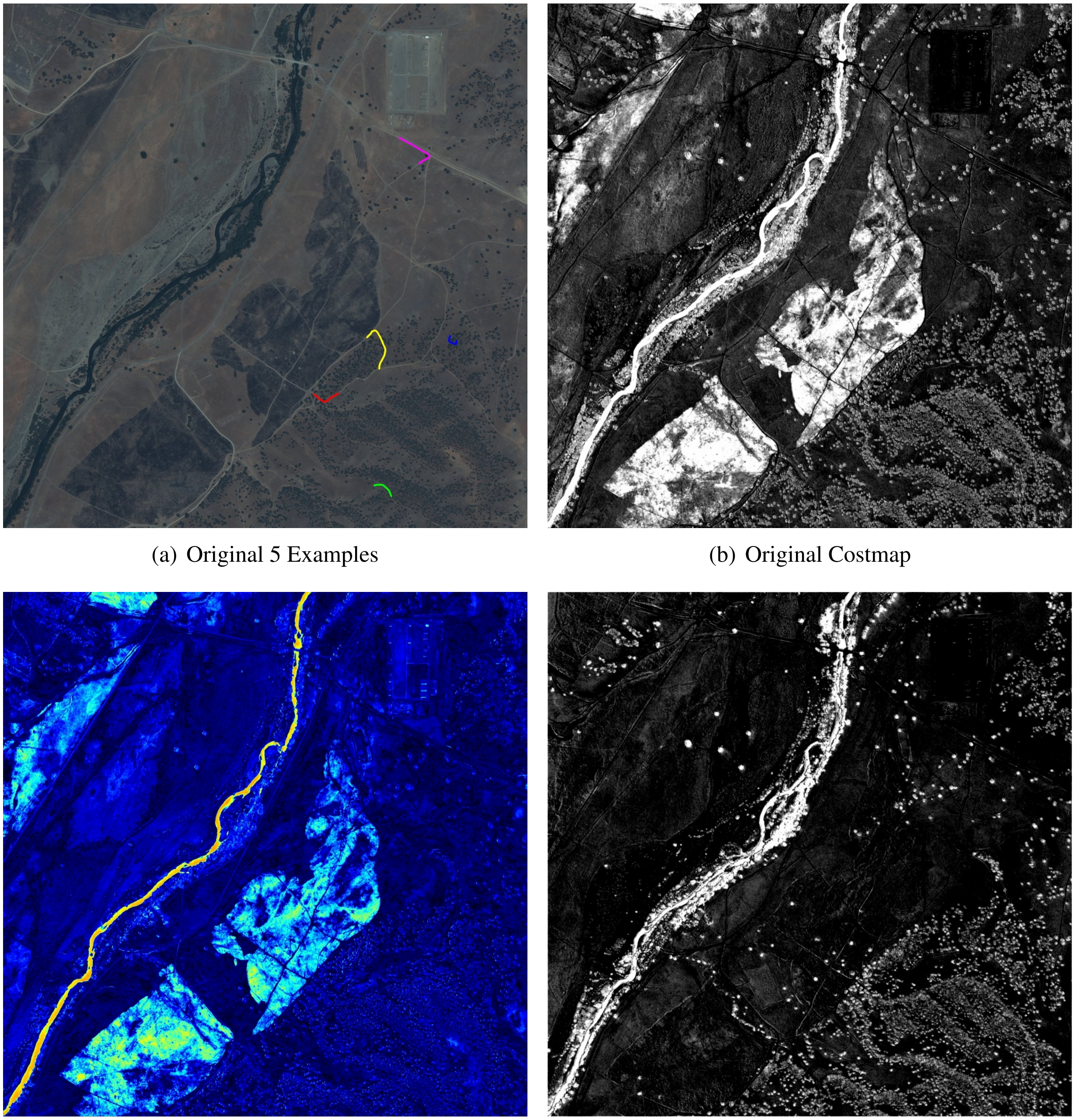


Figure 7.8: By combining acceptable and lethal examples, learning from expert feedback is possible without explicit expert demonstration

(Section 7.1.3) to the training set. Figure 7.7 shows the results of running the original redundancy experiment, both with and without lethal examples. When running with lethals, a very small set of lethal examples was added (at each stage of the experiment). The number of states labeled as lethal was approximately the same number of states as would be expected to be constrained by a single long example path. Since it involved labeling actual locations, and only needed to label obviously lethal sections of terrain, this labeling process was incredibly fast (approximately 1-2 minutes of expert interaction, again on par with adding a single example path). The effect of adding lethal examples is to significantly increase validation performance for smaller training sets. Once the training set becomes sufficiently large to ensure that the same preferences are implied this improvement mostly disappears. However, improved performance over small training sets, as well as a decrease in the variability of this performance (significant at the 1% for training sets with 5 or 10 paths) indicates that the addition of lethal examples can help to improve both stability and consistency, without requiring the expert to resort to providing overly large training sets.

A final experiment was performed as a proof of concept demonstration for the approach described in Section 7.1.4 of collecting expert feedback without any explicit demonstration of desired behavior. Beginning with a blank costmap, the expert chose two waypoints, and rated a series of example paths as unacceptable until an acceptable path was produced. Successive paths were produced by slightly raising the cost along each previous unacceptable path. After each set of acceptable and unacceptable paths was produced, a cost function was learned, and then the entire process repeated. At the very end, lethal examples were also added. Figure 7.8 shows the results of this experiment. The final performance achieved is approximately equivalent to the performance with using half of the original training set in Figure 7.5. This demonstrates that the combination of acceptable and lethal examples, while not the most efficient way of gathering expert feedback, is a feasible approach when actual demonstration is too difficult or costly.



(a) Original 5 Examples

(b) Original Costmap

(c) Novelty Map

(d) Costmap after 5 new examples

Figure 7.9: An example of density based novelty detection and active learning on overhead data.

7.3.2 Active Learning

Novelty based active learning was implemented as described in Section 7.2.1. First, given an existing training set a cost function is learned, and every feature vector encountered during training is recorded. The original feature space consisted of 30 dimensions, an unwieldy number for performing efficient density estimation. Therefore, PCA was used to reduce the data set to 3 dimensions. Once in this space, the K-Means algorithm [278] is used to identify clusters of points and their as-

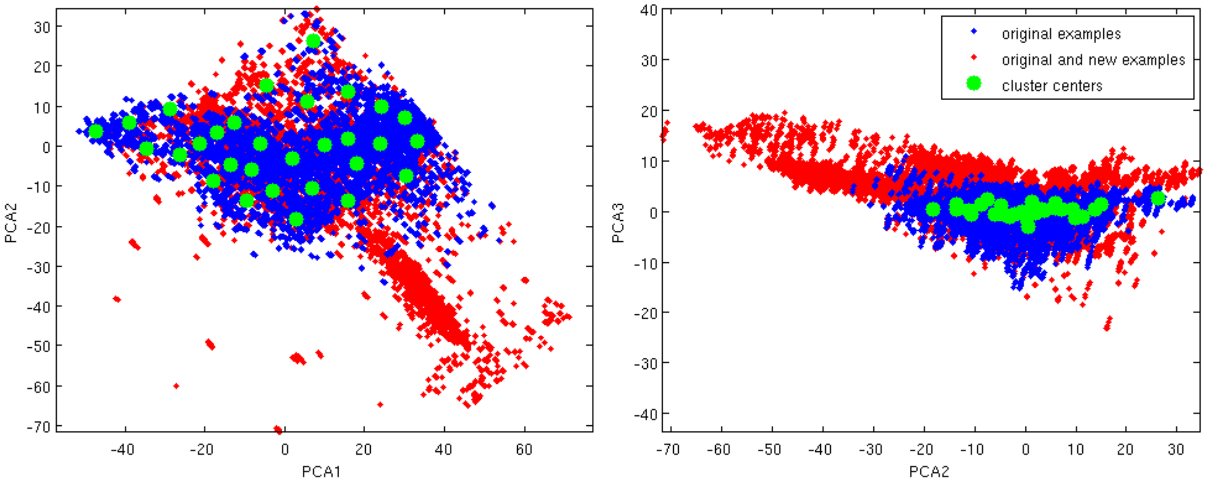


Figure 7.10: Density estimation for novelty detection in Figure 7.9. The full feature space is first reduced to 3 dimensions through PCA. Next, K-Means is used to determine cluster centers to approximate the density of the original distribution. Active learning results in new examples that reside in previously uncovered regions of the space.

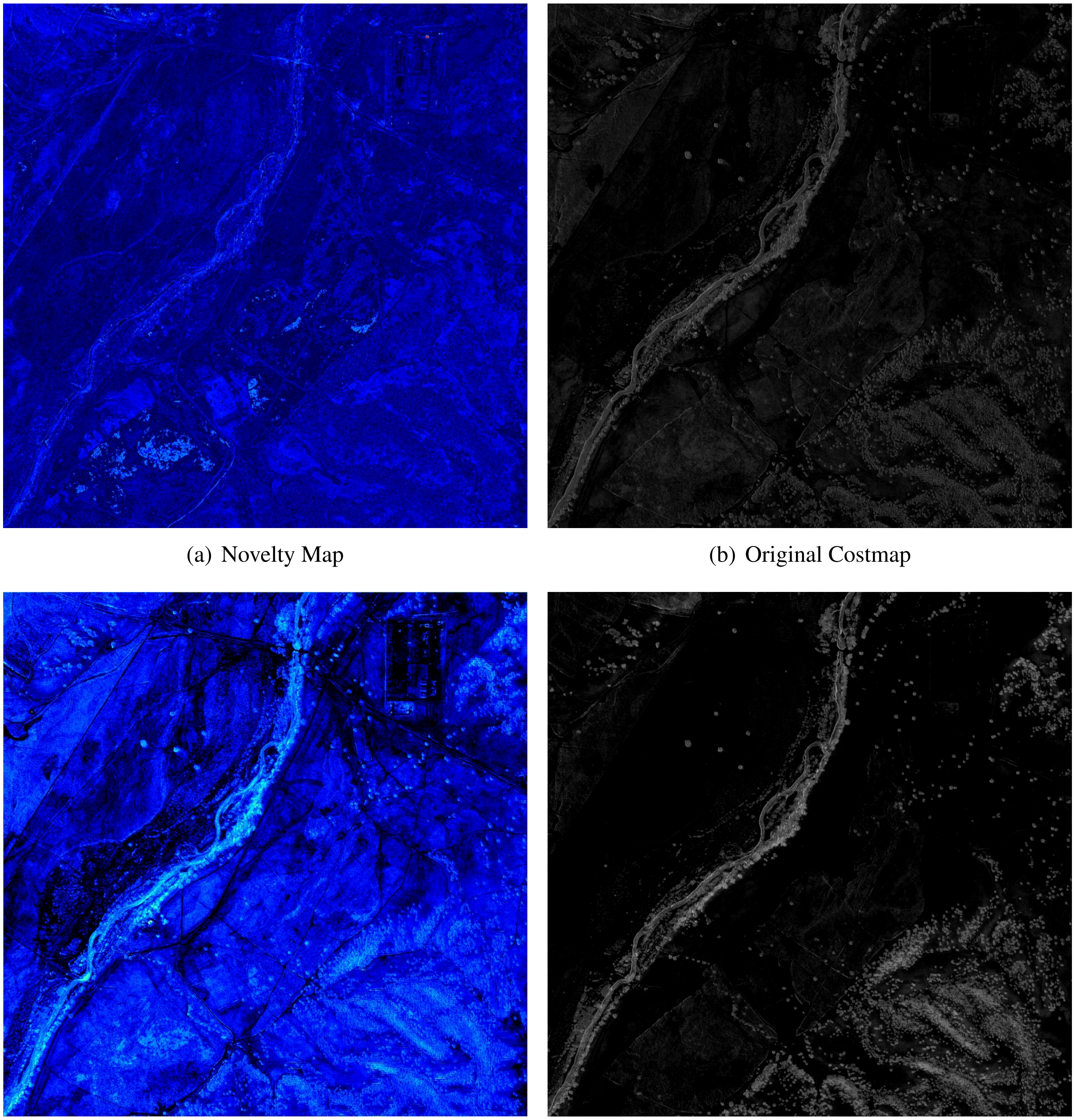
sociated cluster centers. The number of cluster centers K-Means is tasked with finding is set very high[3] as the goal is not to actually identify distinct clusters, but simply regions of the feature space with high point density. For a test point in the reduced space, the novelty is then defined as the squared Euclidean distance to the nearest cluster center. While a very simplistic density estimation and novelty detection approach, it proves more than sufficient for identifying regions of the feature space that have received little to no coverage. More powerful techniques (such as using the cluster centers to seed Gaussian Mixture Models) could be easily inserted if necessary.

Figures 7.9 and 7.10 show a single example iteration of this process. Figure 7.9(a) shows the original 5 example paths. These paths only traverse or avoid roads, tan-colored grass, and trees. As a result, the costmap learned from this training set (Figure 7.9(b)) must try to generalize to other terrain features. However, it does a very poor job of generalizing to where there was a recent controlled burn of the grass (the dark grey regions). These areas are given an extremely high cost, when in fact they are very preferable terrain to traverse through (due to the almost complete lack of vegetation).

After performing density estimation, the novelty map (the novelty function applied to the entire test site) is shown in Figure 7.9(c). As would be expected given the initial training set roads, trees, and grass are low novelty while the burned areas, the river, and buildings are high novelty. 5 new examples were then requested, by picking random waypoints and evaluating the novelty of the current plan. This results in examples that traverse the burned areas, helping to demonstrate their actual preferability. The cost function learned as a result of the new training set (5 original examples and 5 new examples) is shown in Figure 7.9(d), with a more reasonable cost estimate over novel regions.

Figure 7.10 shows the distribution of observed feature vectors (in the PCA reduced space), over both the original and combined training set. The original training set is confined to one region of

[3] in this work 30-50



(a) Novelty Map

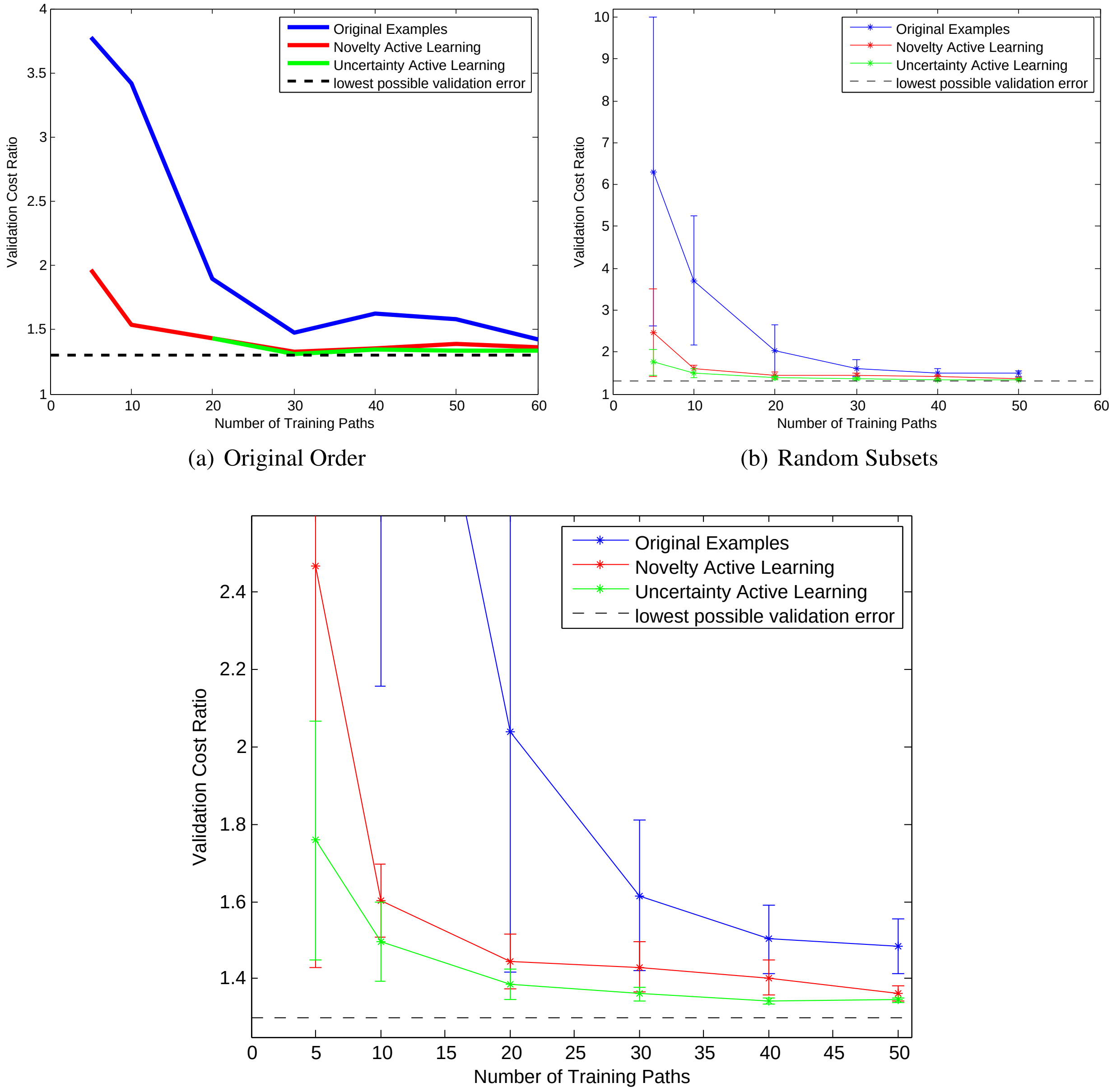
(b) Original Costmap

(c) Uncertainty Map

(d) Costmap after new examples

Figure 7.11: An example of uncertainty based active learning, and how it can complement novelty based active learning.

the feature space (due both to the homogeneous nature of the original set, as well as the reduced space being computed according to its distribution). The cluster centers identified via K-Means are able to represent the distribution of this set with orders of magnitude fewer points. In contrast, the examples selected during active learning result in features that occupy completely different regions of the space. This demonstrates the ability of novelty based active learning to ensure that all regions of the feature space (corresponding to the variety of observed terrain features) are



(a) Original Order

(b) Random Subsets

(c) Random Subsets (Zoomed)

Figure 7.12: Demonstration of how training sets chosen via active learning can perform superior to training sets chosen by an expert

covered via example demonstration.

Uncertainty based active learning was also implemented using Query by Bagging to produce an uncertainty estimate in the cost function. Figure 7.11 provides an example of this approach in action. A training set of 20 example paths were used to train the cost function shown in Figure 7.11(b). This cost function underestimates the cost on several types of undesirable terrain, such as slopes, trees, and water. However, since the original training set did include some examples of these terrain features, they are not considered particularly novel (Figure 7.11(a)). However,

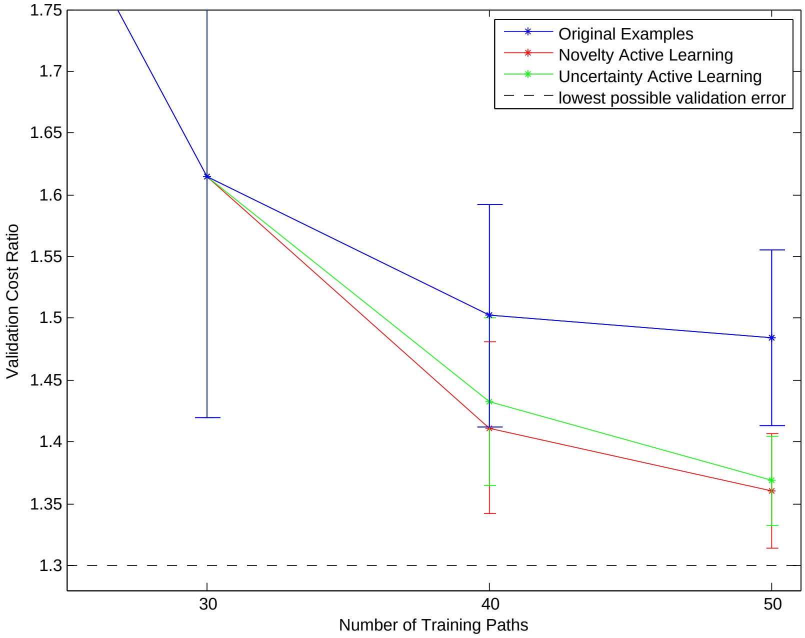


Figure 7.13: Even when a reasonably sized training set already exists, active learning can identify useful and important new examples.

Query by Bagging is able to identify these areas as regions of higher cost uncertainty ((Figure 7.11(c)). That is, when training on some subsets of the full training set, these regions are given a high cost, and on others a lower cost. In contrast, regions that are always given a similar cost (such as roads) are given a very low uncertainty. 10 additional examples chosen via uncertainty based active learning provide additional demonstration of the true preferences over the high uncertainty regions; the resulting cost function (Figure 7.11(d)) reflects this new information via increased cost on trees, water, and slopes. This not only demonstrates the effectiveness of uncertainty based active learning, but also the way in which it can complement novelty based active learning.

The novelty and uncertainty functions in Figures 7.9 and 7.11 also demonstrate that, aside from just being useful for offline (pool-based) active learning, there is potential for online query filtering and vehicle safeguarding. What is especially useful about these functions is that they have been learned not with respect to the underlying perception system, but rather with respect to the underlying cost function. That is, the perception system may be highly confident in its classification and its identification of geometric properties of a patch of terrain, while the cost function is still uncertain about the desirability of the combination of such features. This makes such functions an ideal complement to novelty or uncertainty functions learned for a perception system, and could prove quite useful in the application of such techniques [269].

In addition to proof of concept experiments, larger active learning experiments were also performed to produce quantitative results. In order to compare active learning based example selection to expert based example selection, 60 example paths were chosen using multiple iterations of novelty based active learning (in order to seed the novelty function when the training set is empty, a random subset of the entire training area is used). An additional 40 example paths were chosen using multiple iterations of uncertainty based active learning, with the first 20 novelty chosen paths as a seed (uncertainty based active learning requires an initial training set). The result is two (partially overlapping) sets of training examples that can be compared to the expert chosen set used in the redundancy experiments.

Figure 7.12(a) shows the validation performance of these three training sets when applied in their original order. The first fact of note is that for small training sets, novelty based active learning significantly outperforms expert chosen examples, demonstrating the ability of active learning to identify important novel examples. As more examples are added to each set, the final performance of all 3 sets converges. However, the second fact of note is that both active learning approaches suffer less of a decrease in performance in later stages (before convergence), indicating that examples chosen through active learning are still more likely to contain useful information as opposed to redundant (and possibly inconsistent) information.

Figures 7.12(b) and 7.12(c) show the performance of all 3 training sets when learning with multiple subsets in random order. Both active learning training sets consistently perform better than the expert chosen set (significant at the 1% level). Also of note is that the uncertainty active learning set performs better than the novelty based set in a consistent manner(significant at the 5% level up to 20 examples, and at the 1% level subsequently). Since these sets share one third of their examples, this demonstrates how uncertainty based active learning complements novelty based active learning. Also of note is the variance in performance of each training set. The two active learning approaches achieve a (statistically significant) lower variance than the expert chosen set at each subset size. However, the uncertainty based approach achieves a lower variance than the novelty based approach; this difference is statistically significant (at the 1% level) for subsets of size 30 and greater. This concretely demonstrates the advantage that uncertainty based active learning has over novelty based active learning: it specifically chooses examples that are likely to reduce this variance. Finally, it is informative to observe that the active learning based approaches achieve better performance with fewer examples; The novelty based set has lower performance with 30 or more examples than the expert chosen set with 50 examples, and the same is true with the uncertainty based set for 20 or more examples. Overall, the take away result is that choosing a training set via active learning can achieve better and more consistent performance, with equal or fewer expert demonstrated examples.

In addition to choosing training sets from scratch, active learning can also improve the performance of previously existing training sets. To demonstrate this effect, active learning was used to add additional training data to an expert chosen set. Starting from each different subset of 30 paths of the expert chosen demonstration, two iterations of active learning was performed, each adding 10 new examples. This process was repeated for both novelty and uncertainty based active learning.The results are shown in Figure 7.13, along with the effect of adding additional expert chosen examples. As described before, for expert chosen examples there are diminishing returns; the difference in performance (in a paired manner) is not even statistically significant between 40 and 50 expert chosen examples. In contrast, both active learning approaches result in validation performance that is significantly different (at the 1% level) then the corresponding expert chosen set. This demonstrates that even when efficiency is not an issue, active learning can identify important and useful examples, improving consistency and stability.

Chapter 8

Conclusion

This work has addressed the problem of preference model construction in autonomous mobile robotic systems. Chapters 2 and 3 first described the ways in which preference models have become increasingly important as mobile robots have grown in complexity, along with the environments and scenarios within which they are expected to operate. Prior to this work, the primary approach to developing these models was manual design and tuning. However, such hand tuning approaches require a large amount of complex expert interaction, typically produce suboptimal performance, and must often be repeated from scratch.

Chapter 4 presented various solutions that had previously been applied to simplifying this problem. Although previous applications of machine learning and physical simulation proved useful in making this task tractable, they did not directly address the issue of preference model construction, and still resulted in a manual parameter tuning task, requiring a great deal of interaction from a human expert. However, the framework of Inverse Optimal Control was shown to offer a solution involving the explicit learning of preference models from expert demonstration of desired behavior.

Chapter 5 developed a solution to this problem in the domain of interpreting static perceptual data. The MMP framework was presented along with the LEARCH algorithm, and its applicability demonstrated in this context. The LEARCH algorithm was then extended to the domain of interpreting dynamic perceptual data, resulting in the D-LEARCH algorithm. The issue of imperfect and noisy demonstration was also addressed, resulting in the R-LEARCH and DR-LEARCH algorithms. Experimental results and comparisons to hand tuned preference models were presented from the Crusher robot.

Chapter 6 addressed the problem of driving styles and preferences in a autonomous robot by demonstrating how expert demonstration could be used to learn a cost function over actions as well as terrains. The PD-LEARCH algorithm was developed to account for receding horizon effects as well as noisy and imperfect demonstration. The use of DR-LEARCH and PD-LEARCH to learn coupled preference models from a single set of expert demonstration was also derived, and backed up by experimental results and comparisons to hand tuned models from the E-Gator robot.

Finally, Chapter 7 addressed issues of efficiency and repeatability in learning from demonstration. Alternate forms of interpreting expert demonstration were derived, as well as other constraint based approaches for interpreting expert feedback. These approaches were linked to existing forms of supervised learning, demonstrating a single coherent framework. In addition, two approaches to active learning were developed to aid in the process of determining which demonstrations to provide to a learner. Experiments demonstrated the effectiveness of these techniques in produc-

Application	Performance Improvement	Expert Time Savings
Interpreting Static Overhead Data	Large	˜5x
Interpreting Dynamic Perceptual Data	Small	> 100x
Learning Preferences over Actions	Small	> 6x
Full Mobile Robot IOC	Large	> 14x
Active Learning	Equivalent	˜2x
Active Learning	Small	Equivalent

Table 8.1: A summary of experiments involving components of fielded mobile robotic systems (Sections 5.5 and 6.4) . In each case, there was at least small improvement in performance, as well as a reduction in the required expert interaction by a factor of five. The tasks requiring more expert hand tuning saw a larger time savings. In addition, active learning demonstrated the potential to achieve equivalent performance with half as much expert demonstration, or improved performance with equivalent amounts of demonstration.

ing more stable and consistent training sets, while further reducing the amount of required expert interaction.

8.1 Contributions

The stated goal of this thesis (Section 3.3) was to develop "Principled, automated methods of determining and validating preference model design and parameter settings...that involve at least an order of magnitude less human interaction, while producing equivalent if not better system performance." Table 8.1 provides an overview of experimental results when applying the developed, automated techniques to preference model design. In each experiment, there was at minimum a small improvement in system performance. In addition, there was at minimum a reduction by a factor of five in the required amount of expert interaction. The more complicated and time consuming the original hand tuning task, the more time savings was achieved. In addition, active learning experiments were shown to produce equivalent performance with even less expert interaction, or superior performance with equivalent amounts. Therefore, this work has successfully addressed its initial problem statement.

Chapter 1 specified a set of high level requirements for the design, development, and deployment of mobile robotic systems to continue to move forward. Chapter 3 identified the problem of preference model construction as impeding each of these requirements in one form or another. The solutions provided in this work can contribute to meeting each of these requirements:

Reliable : Easier offline validation can produce more reliable systems. Learning from demonstration allows for easy offline validation of the entire decision making subsystem. This allows online testing to be more about full system validation, and less about shaking out bugs in preference models. In addition it is easier to ensure high reliability under common conditions by simply adding examples of those conditions to the training set. Finally, by using formal methods of preference model construction, the issue of blame assignment is removed; if a cost function simply cannot be learned to reproduce a certain training example, then the problem is not with cost function tuning, but elsewhere in the system (e.g. perception or planning). Such methods also allow for automated solutions to this problem (e.g. a feature learning phase).

Robust Learned cost functions are better regularized than hand tuned ones (through more efficient optimization). As a consequence there is theoretical support for better generalization of learned cost functions, backed up by each experiment performed in this work. There is also less overfitting to specific scenarios at the consequence of others. Online identification of novel scenarios (with regards to previous demonstration) can also improve robustness by avoiding uncertain situations.

Rapid Development By dramatically reducing the amount of expert interaction required to couple components into an operational robotic system, initial deployment is easier. When undesirable behavior is observed in early testing, it is much faster to augment the training set than to manually retune the system. In addition, it is faster to retask a system for completely novel environments, scenarios, or operating conditions. This time savings continues to accrue throughout the life of a robotic system.

Reusable The time saved on preference model construction and tuning can now be better spent on building improved, reusable component technologies. In addition, the effort expended during example demonstration can be reused whenever the system architecture or components are modified (with the exception of certain hardware changes). Depending on the systems in question, it may even be possible to reuse demonstrations intended for one robot to train another (sufficiently similar) robot.

In addition to addressing the core problem statement, this work claims the following additional contributions

- **First application of Inverse Optimal Control towards fully defining the desired behavior of an autonomous mobile robot.** The desired behavior of the E-Gator system was completely defined by the provided training demonstrations. As said demonstrations avoided trees but drove over tall grass, and preferred driving forward to driving in reverse, the same behavior was exhibited by the robot. Had the training demonstrations preferred driving in reverse and avoided even tall grass, the robot would have done the same. No additional changes to the E-Gator system or individual components would have been necessary. In addition, this work has demonstrated solutions to many real world challenges inherent in applying inverse optimal control
  - Collecting and managing training and validation data demonstrated by multiple experts, in separate and varying environments, over a period of months.
  - Interpretation of expert intent from robot sensor logs, and projection of intent into a planner's action space
  - Robustness to noisy, imperfect, and inconsistent expert demonstration.

- **First principled approach for determining terrain costs within autonomous navigation systems.** This work has directly addressed an issue that was inherent to many navigation systems of the last decade, and eloquently expressed in [95]: "Correct system behavior was only obtained by squarely addressing the question of what amount of extra excursion is justifiable to avoid a given level of risk. Nonetheless, we can report few principled methods beyond understanding that the tradeoff exists and tuning the costs in order to get correct behavior under various circumstances."

- **First use of inverse optimal control under dynamic and changing conditions in a robotic system**. The dynamic framework developed in the D-LEARCH, DR-LEARCH and PD-LEARCH algorithms and first presented in [231] represents the first derivation and application of inverse optimal control when the state of the work can evolve in response to anything other than the expert's input. As opposed to a single static plan, this requires the expert to continually re-evaluate his own behavior, and a learning system to interpret expert demonstration in a similar manner.

- **First framework that explicitly learns multiple preference models to produce the correct overall system behavior.** High levels of component performance is a necessary but not sufficient condition for good system performance. Likewise, in a system with multiple preference models it is necessary that they not only express the correct preferences in isolation, but the preferences inherent to each model are properly balanced. This work has demonstrated the ability to learn multiple models from a single training set, tuning both models to produce the desired behavior both in isolation and when combined.

- **Demonstration of the inherent ability of the single arc motion planner to perform complex maneuvers, given the correct preference model.** The single arc motion planner has remained popular due to its inherent simplicity, effectiveness, and ease of implementation. However, in the past a good deal of its power has come from additional layers of planning hand tuned to perform specific maneuvers (e.g. N-point turns) in specific scenarios. This work has demonstrated that these additional layers are unnecessary for a wide range of complex maneuvers, if the action preference model is properly tuned.

- **Demonstration of an approach and metrics for offline validation of mobile robot performance**. The necessary machinery for performing learning from demonstration provides a natural solution to the problem of validating actual robot behavior offline. Such an approach is useful even if learning from demonstration is not explicitly used, as are the metrics used to evaluate such behavior and its similarity to desired behavior.

- **Experimental demonstration of the performance improvement and time savings when using the developed learning from demonstration algorithms.** This work has demonstrated not only the feasibility of learning preference models from expert demonstration, but has produced quantitative results from multiple experiments demonstrating significant performance improvements and time savings over previous hand tuning approaches (Table 8.1). The initial time savings demonstrated by these experiments will only increase as each system evolves (and no further expert tuning is required). As a consequence, this work claims to have demonstrated the best solution to date for the construction of preference models in mobile robotic systems.

- **Demonstration of the relationship between constraint based interpretations of expert demonstration and feedback.** This work has not only developed several techniques for learning preference models from demonstration, but has also shown how they can be combined with other forms of expert supervision [159, 160, 74, 77, 79, 99, 80, 83, 84, 102] and preference solicitation [252, 253, 254] into a single, consistent framework that considers all expert input in the context of the behavior it suggests.

- **Development of active learning techniques appropriate to generalizable learning from demonstration.** This work has presented two general approaches to applying active learning to learning from demonstration, specifically in the setting when the learner can generalize from well chosen, important examples. These approaches have both been demonstrated to produce more stable and consistent training sets when used offline in a pool based manner; equivalently they can also be used to minimize the required expert interaction. In addition, both approaches provide the necessary mechanisms to allow for either additional online vehicle safeguarding, or query filtering approaches for supervised autonomy.

8.2 Future Work

The development of fieldable mobile robotic systems often contains many problems whose solutions rely on human intuition, guesswork, and trial-and-error or generate-and-test procedures. The design and tuning of preference models was one such situation, an issue which this thesis has directly addressed. Another instance of this form of problem is often in the production of training sets for a myriad of learning components: a large set of examples is produced, and then it is continually tweaked and added to as incorrect or undesirable results are observed. The application of active learning techniques presented in this thesis is one approach for the reducing burden on expert intuition in this process. Automatic reweighting of a training set to match the observed distribution of terrain features in a specific environment, proposed in [279] for terrain classification, is another idea that merits future research, both as an offline and online approach.

One area that still requires a good deal of both expert intuition and trial-and-error is in feature design. For instance, in the perceptual domain, there are enormous bodies of work investigating useful and efficient feature extraction, both from a signal processing as well as a biologically inspired perspective. The use of machine learning techniques allows for automated interpretation of these features, and sparse learning can help to identify and remove useless or redundant features. However, if the standard set of off-the-shelf feature extractors proves insufficient, the next step is often specialized, manually engineered feature extractors.

Chapter 5 presented a feature learning phase as potential solution to this problem; however this only works if a nonlinear combination of existing feature is sufficient. When this proves insufficient, what is required is to backpropagate the learning signal (based on expert desired behavior) even further. [279, 280] have explored this idea when there is a learning (i.e. classification) layer between perceptual features and a perceptual cost function. However, another idea is to backpropagate the learning signal all the way to the sensors themselves [281]. For example, if a perceptual cost function was unable to distinguish between certain high cost and low cost terrain (i.e. a specific demonstrated example could not be learned) the states corresponding to both the example and current planned paths could be projected into the raw perceptual space (e.g. camera images) and new feature extractors learned at an earlier level. Such an approach would allow robotic systems to continue to make use of principled, general, and previously implemented feature extraction and signal processing techniques, while removing the extra engineering effort necessary to design specialized extractors for specific environments and scenarios (which may not have further use outside of their intended scenario).

Another avenue for future investigation is motion planners with longer planning horizons. The single arc motion planner was used in this work due to its balance of simplicity and effectiveness. Given that the result was a planning system which could produce complex maneuvers that it did not explicitly plan (that is, the maneuver 'fell out' of repeated decision making), the application to kinodynamic motion planners that search multiple actions for longer horizons can be expected to, at a minimum, similarly reproduce an expert's preferences for certain driving styles and maneuvers. However, it is unclear which robust version of LEARCH, PD-LEARCH or DR-LEARCH would be most effective in such a context at accounting for noisy and imperfect demonstration; a combination of the two could potentially prove most effective. It is also possible that an entirely different class of penalty functions would be useful or necessary. Such planning systems also allow for the possibility of explicitly learning preferences with regards to velocity control.

Of course, with an issue such as velocity control it is not clear that matching an expert's preferences is the proper way to make such decisions. In this case, it can be argued that it is better to determine through simulation and online learning [192, 193] how fast the robot can safely travel, and then execute an expert desired plan at safe speed. However, it is possible that the expert's desires may depend on his notion of the safest possible speed; it is also possible that the robot may have a better prediction of this speed than the expert. This demonstrates just one of the many instances where the ideas of online learning, physical simulation, and learning from demonstration can all be applied. In the future, one can imagine mobile robotic systems that learn from experience to predict terramechanics and mobility properties of different terrains, use physical simulation to understand the consequences of various actions over these terrains, and use expert demonstration to learn preferences over the consequences of these actions. Such a fusion of these ideas offers the promise of further improving autonomous reliability and robustness in both familiar and novel environments, while reducing both initial and continual development efforts, thus bringing us closer to the fast, easy, and wide application of autonomous mobile robotic systems.

Acknowledgements

This work was sponsored by DARPA under contract "Unmanned Ground Combat Vehicle - PerceptOR Integration" (contract MDA972-01-9-0005), the U.S. Army Research Laboratory under contract "Robotics Collaborative Technology Alliance" (contract DAAD19-01-2-0012), and TARDEC under contract "Incremental Learning for Robot Sensing and Control" (contract W56HZV-10-C-0176). The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

This work required numerous field experiments on the Crusher and E-Gator robots, over a period of several years. These experiments would not have been remotely feasible except for an incalculable amount of support, both in system development and field logistics. I would like to thank the entire UPI team of the CMU National Robotics Engineering Center and the rCommerce team of the CMU Field Robotics Center for their efforts and patience in this work; it would not have been possible without your help.

Bibliography

[1] H. F. Durrant-Whyte, “An Autonomous Guided Vehicle for Cargo Handling Applications,” *The International Journal of Robotics Research*, vol. 15, no. 5, pp. 407–440, 1996. 10

[2] E. Duff, J. Roberts, and P. Corke, “Automation of an underground mining vehicle using reactive navigation and opportunistic localization,” in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 4, pp. 3775–3780, Oct. 2003. 10

[3] M. Dunbabin and P. Corke, “Autonomous excavation using a rope shovel,” *Journal of Field Robotics*, vol. 23, pp. 379–394, 2006. 10

[4] H. Schempf, E. Mutschler, V. Goltsberg, G. Skoptsov, A. Gavaert, and G. Vradis, “Explorer: Untethered real-time gas main assessment robot system,” in *Proc. of Int. Workshop on Advances in Service Robotics*, 2003. 10

[5] A. Morris, D. Ferguson, Z. Omohundro, D. Bradley, D. Silver, C. Baker, S. Thayer, W. Whittaker, and W. L. Whittaker, “Recent developments in subterranean robotics,” *Journal of Field Robotics*, vol. 23, pp. 35–57, January 2006. 10

[6] J. Bares and D. Wettergreen, “Dante ii: Technical description, results, and lessons learned,” *International Journal of Robotics Research*, vol. 18, pp. 621–649, July 1999. 10

[7] D. Wettergreen, C. Gaskett, and A. Zelinsky, “Autonomous guidance and control for an underwater robotic vehicle,” in *Field and Service Robotics*, 1999. 10, 41

[8] T. Estlin, B. Bornstein, D. Gaines, D. R. Thompson, R. Castano, R. Anderson, C. de Granville, M. Burl, M. Judd, and S. Chien, “Aegis automated targeting for the mer opportunity rover,,” in *International Symposium on Artificial Intelligence, Robotics and Automation in Space (*, 2010. 10

[9] P. Singer, *Wired for War*. Penguin Press, 2009. 10

[10] N. Ratliff, D. Silver, and J. A. Bagnell, “Learning to search: Functional gradient techniques for imitation learning,” *Autonomous Robots*, 2009. 16, 45, 46, 52, 53

[11] H. Moravec, *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. PhD thesis, Stanford, September 1980. 16

[12] H. Moravec, “The stanford cart and the cmu rover,” *Proceedings of the IEEE*, vol. 71, pp. 872–884, July 1983. 16

[13] Champeny-Bares, L. S. Coppersmith, and K. Dowling, "The terregator mobile robot," Tech. Rep. CMU-RI-TR-93-03, Robotics Institute, Pittsburgh, PA, May 1991. 16

[14] R. Wallace, A. Stentz, C. Thorpe, H. Moravec, W. L. Whittaker, and T. Kanade, "First results in robot road-following," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1985. 17

[15] M. Hebert and T. Kanade, "Outdoor scene analysis using range data," in *Proc. 1986 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1426–1432, April 1986. 17

[16] T. Kanade, C. Thorpe, and W. L. Whittaker, "Autonomous land vehicle project at cmu," in *CSC '86: Proceedings of the 1986 ACM fourteenth annual conference on Computer science*, (New York, NY, USA), pp. 71–80, ACM, 1986. 17

[17] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the carnegie-mellon navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, pp. 362 – 373, May 1988. 17

[18] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Toward autonomous driving: The cmu navlab. part i: Perception," *IEEE Expert*, vol. 6, pp. 31 – 42, August 1991. 17

[19] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Toward autonomous driving: The cmu navlab. part ii: System and architecture," *IEEE Expert*, vol. 6, pp. 44 – 52, August 1991. 17

[20] E. Dickmanns, R. Behringer, D. Dickmanns, T. Hildebrand, M. Maure, F. Thomanek, and J. Schiehlen, "The seeing passenger car vamors-p," in *Intelligent Vehicles Symposium*, pp. 68–73, October 1994. 17

[21] R. Behringer and N. Muller, "Autonomous road vehicle guidance from autobahnen to narrow curves," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 810–815, October 1998. 17

[22] J. Bishop, M. Juberts, and D. Raviv, "Autonomous vision-based technology for avcs," in *Vehicular Technology Conference*, pp. 360–363, May 1993. 17

[23] D. Pomerleau and T. Jochem, "Rapidly adapting machine vision for automated vehicle steering," *IEEE Expert: Special Issue on Intelligent System and their Applications*, vol. 11, pp. 19–27, April 1996. 17

[24] T. Jochem and D. Pomerleau, "Life in the fast lane:the evolution of an adaptive vehicle control system," *AI Magazine*, vol. 17, no. 2, pp. 11–50, 1996. 17

[25] E. Dickmanns, "The development of machine vision for road vehicles in the last decade," in *IEEE Intelligent Vehicle Symposium*, vol. 1, pp. 268–281, June 2002. 17

[26] U. Franke, D. Gavrila, S. Gorzig, F. Lindner, F. Puetzold, and C. Wohler, "Autonomous driving goes downtown," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 6, pp. 40–48, 1998. 17

[27] L. Zhao and C. Thorpe, “Stereo and neural network-based pedestrian detection,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, pp. 148–154, September 2000. 17

[28] C.-C. Wang, C. Thorpe, and A. Suppe, “Ladar-based detection and tracking of moving objects from a ground vehicle at high speeds,” in *IEEE Intelligent Vehicles Symposium*, June 2003. 17

[29] C. Thorpe, J. D. Carlson, D. Duggins, J. Gowdy, R. MacLachlan, C. Mertz, A. Suppe, and C.-C. Wang, “Safe robot driving in cluttered environments,” in *International Symposium of Robotics Research*, 2003. 17

[30] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, “Stanley: The robot that won the darpa grand challenge,” *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006. 18, 37, 39, 53

[31] C. Urmson, C. Ragusa, D. Ray, J. Anhalt, D. Bartz, T. Galatali, A. Gutierrez, J. Johnston, S. Harbaugh, H. ldquo, Y. Kato, W. Messner, N. Miller, K. Peterson, B. Smith, J. Snider, S. Spiker, J. Ziglar, W. L. Whittaker, M. Clark, P. Koon, A. Mosher, and J. Struble, “A robust approach to high-speed navigation for unrehearsed desert terrain,” *Journal of Field Robotics*, vol. 23, no. 8, pp. 467–508, 2006. 18, 56

[32] P. G. Trepagnier, J. Nagel, P. M. Kinney, C. Koutsougeras, and M. Dooner, “Kat-5: Robust systems for autonomous vehicle navigation in challenging and unknown terrain,” *Journal of Field Robotics*, vol. 23, no. 8, pp. 509–526, 2006. 18

[33] D. Braid, A. Broggi, and G. Schmiedel, “The terramax autonomous vehicle,” *Journal of Field Robotics*, vol. 23, no. 9, pp. 693–708, 2006. 18

[34] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski, “Self-supervised monocular road detection in desert terrain,” in *Proceedings of Robotics: Science and Systems*, (Philadelphia, USA), August 2006. 18, 39

[35] M. Buehler, “Summary of dgc 2005 results,” *Journal of Field Robotics*, vol. 23, pp. 465–466, 2006. 18

[36] C. Urmson, J. Anhalt, J. A. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. L. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, “Autonomous driving in urban environments: Boss and the urban challenge,” *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008. 19, 27, 30, 89, 90

[37] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun, “Junior: The stanford entry in the urban challenge,” *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2208. 19, 30, 69, 89

[38] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson, S. Cacciola, P. Currier, A. Dalton, J. Farmer, J. Hurdus, S. Kimmel, P. King, A. Taylor, D. V. Covern, and M. Webster, “Odin: Team victortango’s entry in the darpa urban challenge,” *Journal of Field Robotics*, vol. 25, no. 8, pp. 467–492, 2008. 19, 30

[39] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams, “A perception-driven autonomous urban vehicle,” *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008. 19, 30

[40] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, “Little ben: The ben franklin racing team’s entry in the 2007 darpa urban challenge,” *Journal of Field Robotics*, vol. 25, no. 9, pp. 598–614, 2008. 19, 30

[41] I. Miller, M. Campbell, D. Huttenlocher, F.-R. Kline, A. Nathan, S. Lupashin, J. Catlin, B. Schimpf, P. Moran, N. Zych, E. Garcia, M. Kurdziel, and H. Fujishima, “Team cornell’s skynet: Robust perception and planning in an urban environment,” *Journal of Field Robotics*, vol. 25, no. 8, pp. 493–527, 2008. 19, 30

[42] M. Daily, J. Harris, D. Keirsey, D. Olin, D. Payton, K. Reiser, J. Rosenblatt, D. Tseng, and V. Wong, “Autonomous cross-country navigation with the alv,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, pp. 718–726, April 1988. 20, 35, 89

[43] K. Olin and D. Tseng, “Autonomous cross-country navigation: an integrated perception and planning system,” *IEEE Expert*, vol. 6, pp. 16–30, Aug 1991. 20, 35, 89

[44] D. Payton, J. Rosenblatt, and D. Keirsey, “Plan guided reaction,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 20, pp. 1370–1382, Nov/Dec 1990. 20

[45] J. Mitchell, D. Payton, and D. Keirsey, “Planning and reasoning for autonomous vehicle control,” *International Journce of Intelligent Systems*, vol. 2, no. 2, pp. 129–198, 1987. 20, 23, 30

[46] J. Mitchell and C. Papadimitriou, “The weighted region problem,” in *Proceedings of the third annual symposium on Computational geometry*, (New York, NY, USA), pp. 30–38, ACM, 1987. 20

[47] J. Mitchell, “An algorithmic approach to some problems in terrain navigation,” Tech. Rep. TR000779, Cornell University, Feb 1988. 20

[48] E. Mettala, “Reconnaissance, surveillance and target acquisition research for the unmanned ground vehicle program,” in *Proceedings Image Understanding Workshop*, (Washington D.C.), 1993. 20

[49] L. Matthies, “Stereo vision for planetary rovers: Stochastic modeling to near real-time implementation,” *International Journal of Computer Vision*, vol. 8, no. 1, pp. 71–91, 1992. 21

[50] P. Grandjean and L. Matthies, “Perception control for obstacle detection by a cross-country rover,” in *Proceedings IEEE International Conference on Robotics and Automation*, vol. 2, pp. 20–27, May 1993. 21

[51] L. Matthies and P. Grandjean, “Stochastic performance, modeling and evaluation of obstacle detectability with imaging range sensors,” *IEEE Transactions on Robotics and Automation*, vol. 10, pp. 783–792, Dec 1994. 21

[52] M. Hebert, C. Thorpe, and A. Stentz, *Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon*. KluwerAcademic Publishers, 1997. 21, 30

[53] D. Langer, J. Rosenblatt, and M. Hebert, “A behavior-based system for off-road navigation,” *Robotics and Automation, IEEE Transactions on*, vol. 10, pp. 776–783, Dec 1994. 21, 30, 43

[54] D. Langer, J. Rosenblatt, and M. Hebert, “An integrated system for autonomous off-road navigation,” in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 414–419 vol.1, May 1994. 21, 30, 43

[55] A. Kelly, A. Stentz, and M. Hebert, “Terrain map building for fast navigation on rugged outdoor terrain,” in *Proceedings of SPIE Symposium on Mobile Robots*, 1992. 21

[56] J. Rosenblatt and C. Thorpe, “Combining multiple goals in a behavior-based architecture,” in *International Conference on Intelligent Robots and Systems*, 1995. 21, 30

[57] A. Stentz and M. Hebert, “A complete navigation system for goal acquisition in unknown environments,” *Autonomous Robots*, vol. 2, no. 2, 1995. 21, 30

[58] A. Stentz, “Optimal and efficient path planning for partially-known environments,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3310 – 3317, May 1994. 21

[59] A. Kelly, *An Intelligent Predictive Control Approach to the High-Speed Cross-Country Autonomous Navigation Problem*. PhD thesis, Robotics Institute, Carnegie Mellon University, September 1995. 21, 35, 72

[60] A. Kelly and A. Stentz, “Rough terrain autonomous mobility - part 2: An active vision, predictive control,” *Autonomous Robots*, pp. 163 – 198, May 1998. 21

[61] J. R. Spofford, R. D. Rimey, and S. H. Munkeby, “Overview of the ugv / demo ii program,” in *Reconnaissance, Surveillance, and Target Acquisition for the Unmanned Ground Vehicle: Providing Surveillance ”Eyes” for an Autonomous Vehicle* (Firshein and Strat, eds.), Morgan Kaufmann, 1997. 21

[62] C. Shoemaker and J. Bornstein, “The demo iii ugv program: a testbed for autonomous navigation research,” in *Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings*, pp. 644–651, Sep 1998. 21

[63] A. Lacaze, Y. Moscovitz, N. Declaris, and K. Murphy, “Path planning for autonomous vehicles driving over rough terrain,” in *IEEE ISIC/CIRA/ISAS Joint Conference*, pp. 50–55, 1998. 22, 30

[64] T. Hong, S. Legowik, and M. Nashman, “Obstacle detection and mapping,” tech. rep., National Institute of Standards and Technology, 1998. 22

[65] T. Chang, T. Hong, S. Legowik, and M. Abrams, “Concealment and obstacle detection for autonomous driving,” in *Proceedings of the International Association of Science and Technology for Development -Robotics & Applications Conference*, 1999. 22

[66] K. Murphy, M. Abrams, D. Coombs, T. Hong, S. Legowik, T. Chang, and A. Lacaze, “Intelligent control for unmanned vehicles,” in *in Proceeding of the 2000 World Automation Congress Conference*, pp. 11–16, 2000. 22, 30

[67] D. Coombs, K. Murphy, A. Lacaze, and S. Legowik, “Driving autonomously off-road up to 35 km/h,” in *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*, pp. 186–191, 2000. 22

[68] A. Lacaze, K. Murphy, and M. Delgiorno, “Autonomous mobility for the demo iii experimental unmanned vehicles,” in *in Assoc. for Unmanned Vehicle Systems Int. Conf. on Unmanned Vehicles (AUVSI 02*, 2002. 22, 29

[69] J. Albus, K. Murphy, A. Lacaze, S. Legowik, S. Balakirsky, T. Hong, M. Shneier, and E. Messina, “4d/rcs sensory processing and world modeling on the demo iii experimental unmanned ground vehicles,” in *Proceedings of the IEEE International Symposium on Intelligent Control*, pp. 885–890, 2002. 22

[70] S. Balakirsky and A. Lacaze, “Value-driven behavior generation for an autonomous mobile,” in *in Proceedings of the SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, 2002. 22, 30

[71] S. Balakirsky and A. Lacaze, “World modeling and behavior generation for autonomous ground vehicle,” in *Proceedings IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1201–1206, 2000. 22, 28, 30

[72] L. Matthies, T. Litwin, K. Owens, A. Rankin, K. Murphy, D. Coombs, J. Gilsinn, T. Hong, S. Legowik, M. Nashman, and Billibon, “Performance evaluation of ugv obstacle detection with ccd/flir stereo vision and ladar,” in *IEEE ISIC/CIRA/ISAS Joint Conference*, 1998. 22

[73] P. Bellutta, R. Manduchi, L. Matthies, K. Owens, and A. Rankin, “Terrain perception for demo iii,” in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 326–331, 2000. 22

[74] J. Macedo, R. Manduchi, and L. Matthies, “Ladar-based discrimination of grass from obstacles for autonomous navigation,” in *International Symposium on Experimental Robotics*, 2000. 22, 37, 153

[75] A. Talukder, R. Manduchi, R. Castano, K. Owens, L. Matthies, A. Castano, and R. Hogg, “Autonomous terrain characterisation and modelling for dynamic control of unmanned vehicles,” in *IEEE/RSJ International Conference on Intelligent Robots and System*, vol. 1, pp. 708–713, 2002. 22, 37

[76] A. Talukder, R. Manduchi, A. Rankin, and L. Matthies, “Fast and reliable obstacle detection and segmentation for cross-country navigation,” in *IEEE Intelligent Vehicles Symposium*, pp. 610–618, 2002. 22, 37

[77] R. Manduchi, A. Castano, A. Talukder, and L. Matthies, “Obstacle detection and terrain classification for autonomous off-road navigation,” *Autonomous Robots*, vol. 18, pp. 81–102, 2005. 22, 37, 153

[78] N. Vandapel, D. Huber, A. Kapuria, and M. Hebert, “Natural terrain classification using 3-d ladar data,” in *IEEE International Conference on Robotics and Automation*, vol. 5, pp. 5117 – 5122, April 2004. 22, 37

[79] J.-F. Lalonde, N. Vandapel, D. Huber, and M. Hebert, “Natural terrain classification using three-dimensional ladar data for ground robot mobility,” *Journal of Field Robotics*, vol. 23, pp. 839–861, October 2006. 22, 23, 37, 153

[80] C. Dima, N. Vandapel, and M. Hebert, “Classifier fusion for outdoor obstacle detection,” in *International Conference on Robotics and Automation*, vol. 1, pp. 665 – 671, IEEE, April 2004. 22, 37, 153

[81] D. Munoz, N. Vandapel, and M. Hebert, “Onboard contextual classification of 3-d point clouds with learned high-order markov random fields,” in *IEEE International Conference on Robotics and Automation*, May 2009. 23

[82] DARPA, “Program solicitation perception for off-road mobility,” in *DARPA PS01-02*, (Washington D.C.), 2000. 23

[83] L. Matthies, C. Bergh, A. Castano, and J. Macedo, “Obstacle detection in foliage with ladar and radar,” in *In International Symposium on Robotic Research*, pp. 291–300, 2003. 23, 37, 153

[84] D. Bradley, S. Thayer, A. Stentz, and P. Rander, “Vegetation detection for mobile robot navigation,” Tech. Rep. CMU-RI-TR-04-12, Carnegie Mellon Robotics Institute, Pittsburgh, PA, February 2004. 23, 37, 153

[85] L. Matthies, P. Bellutta, and M. McHenry, "Detecting water hazards for autonomous off-road navigation," in *Proceedings of SPIE Conference 5083: Unmanned Ground Vehicle Technology V*, pp. 263–352, 2003. 23

[86] A. Huertas, L. Matthies, and A. Rankin, "Stereo-based tree traversability analysis for autonomous off-road navigation," in *Seventh IEEE Workshops on Application of Computer Vision*, vol. 1, pp. 210–217, Jan. 2005. 23, 30

[87] L. Matthies and A. Rankin, "Negative obstacle detection by thermal signature," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, pp. 906–913 vol, Oct. 2003. 23

[88] A. Stentz, A. Kelly, H. Herman, P. Rander, O. Amidi, and R. Mandelbaum, "Integrated air/ground vehicle system for semi-autonomous off-road navigation," in *Proc. of AUVSI Unmanned Systems Symposium*, July 2002. 23

[89] A. Rieder, B. Southall, G. Salgian, R. Mandelbaum, H. Herman, P. Rander, and A. Stentz, "Stereo perception on an off-road vehicle," in *IEEE Intelligent Vehicle Symposium*, vol. 1, pp. 221–226, June 2002. 23

[90] A. Ansar, A. Huertas, L. Matthies, and S. Goldberg, "Enhancement of stereo at range discontinuities," in *SPIE Unmanned Ground Vehicle Technology VI*, 2004. 23

[91] A. Rankin, A. Huertas, and L. Matthies, "Evaluation of stereo vision obstacle detection algorithms for off-road autonomous navigation," in *UVSI's Unmanned Systems North America*, 2005. 23

[92] K. C. Kluge and M. K. Morgenthaler, "Multi-horizon reactive and deliberative path planning for autonomous cross-country navigation," in *SPIE Unmanned Ground Vehicle Technology VI*, 2004. 23, 29, 30

[93] A. Rankin, C. Bergh, S. Goldberg, P. Bellutta, A. Huertas, and L. Matthies, "Passive perception system for day/night autonomous off-road navigation," in *SPIE Defense & Security Symposium: Unmanned Ground Vehicle Technology VII*, 2005. 23

[94] A. Stentz, A. Kelly, P. Rander, H. Herman, O. Amidi, R. Mandelbaum, G. Salgian, and J. Pedersen, "Real-time, multi-perspective perception for unmanned ground vehicles," in *Proc. of AUVSI Unmanned Systems Symposium*, July 2003. 23

[95] A. Kelly, A. Stentz, O. Amidi, M. Bode, D. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski, P. Rander, S. Thayer, N. Vallidis, and R. Warner, "Toward reliable off road autonomous vehicles operating in challenging environments," *International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 449–483, 2006. 23, 27, 29, 30, 32, 37, 69, 70, 72, 90, 151

[96] E. Krotkov, S. Fish, L. Jackel, B. McBride, M. Perschbacher, and J. Pippine, "The darpa perceptor evaluation experiments," *Autonomous Robots*, vol. 22, no. 1, pp. 19–35, 2007. 23

[97] N. Vandapel, R. R. Donamukkala, and M. Hebert, "Experimental results in using aerial ladar data for mobile robot navigation," in *International Conference on Field and Service Robotics*, 2003. 23, 37

[98] N. Vandapel, R. R. Donamukkala, and M. Hebert, "Quality assessment of traversability maps from aerial lidar data for an unmanned ground vehicle," in *Proceedings of the IEEE/JRS International Conference on Intelligent Robots and Systems*, October 2003. 23, 37

[99] N. Vandapel, R. R. Donamukkala, and M. Hebert, "Unmanned ground vehicle navigation using aerial ladar data," *The International Journal of Robotics Research*, vol. 25, pp. 31–51, January 2006. 23, 37, 153

[100] J. Bares and D. Stager, "Expanded field testing results from spinner, a high mobility hybrid ugcv," in *Proceedings of the AUVSI Unmanned Systems Conference*, 2004. 23

[101] A. Stentz, J. Bares, T. Pilarski, and D. Stager, "The crusher system for autonomous navigation," in *AUVSIs Unmanned Systems*, August 2007. 23, 29, 32, 37, 69, 70, 78, 81, 90

[102] J. A. Bagnell, D. Bradley, D. Silver, B. Sofman, and A. Stentz, "Learning for autonomous navigation: Advances in machine learning for rough terrain mobility," *IEEE Robotics & Automation Magazine*, vol. 17, pp. 74–84, June 2010. 23, 29, 32, 37, 69, 70, 78, 81, 90, 153

[103] B. Sofman, E. Lin, J. A. Bagnell, N. Vandapel, and A. Stentz, "Improving robot navigation through self-supervised online learning," in *Proceedings of Robotics: Science and Systems*, (Philadelphia, USA), August 2006. 23, 39, 83

[104] B. Sofman, E. L. Ratliff, J. A. Bagnell, J. Cole, N. Vandapel, and A. Stentz, "Improving robot navigation through self-supervised online learning," *Journal of Field Robotics*, vol. 23, December 2006. 23, 39, 83

[105] D. Silver, B. Sofman, N. Vandapel, J. A. Bagnell, and A. Stentz, "Experimental analysis of overhead data processing to support long range navigation," in *Proceedings of the IEEE/JRS International Conference on Intelligent Robots and Systems*, October 2006. 24, 37, 56, 75, 76, 78

[106] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila, "Autonomous rover navigation on unknown terrains: Functions and integration," *International Journal of Robotics Research*, vol. 21, pp. 917–942, October 2002. 24, 30

[107] P. Tompkins, A. Stentz, and D. Wettergreen, "Global path planning for mars rover exploration," in *Proceedings of the 2004 IEEE Aerospace Conference*, March 2004. 24, 28, 30

[108] P. Tompkins, A. Stentz, and D. Wettergreen, "Mission-level path planning and re-planning for rover exploration," *Robotics and Autonomous Systems*, vol. 54, pp. 174 – 183, February 2006. 24, 28, 30

[109] R. Castano, M. Judd, R. Anderson, and T. Estlin, “Machine learning challenges in mars rover traverse science,” in *ICML Workshop on Machine Learning Technologies for Autonomous Space*, 2003. 24, 30

[110] T. Smith, S. Niekum, D. R. Thompson, and D. Wettergreen, “Concepts for science autonomy during robotic traverse and survey,” in *IEEE Aerospace Conference*, March 2005. 24, 30

[111] R. Castano, T. Estlin, D. Gaines, C. Chouinard, B. Bomstein, R. Anderson, M. Burl, D. R. Thompson, A. Castano, and M. Judd, “Onboard autonomous rover science,” in *IEEE Aerospace Conference*, pp. 1–13, March 2007. 24, 30

[112] D. R. Thompson and D. Wettergreen, “Intelligent maps for autonomous kilometer-scale science survey,” in *International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*, February 2008. 24, 30

[113] M. Woods, A. Shaw, D. Barnes, D. Price, D. Long, and D. Pullan, “Autonomous science for an exomars rover-like mission,” *Journal of Field Robotics*, vol. 26, pp. 358–390, 2009. 24, 30

[114] L. Matthies, E. Gat, R. Harrison, B. Wilcox, R. Volpe, and T. Litwin, “Mars microrover navigation: Performance evaluation and enhancement,” *Autonomous Robots*, vol. 2, pp. 291–311, 1995. 24

[115] A. Mishkin, J. Morrison, T. Nguyen, H. Stone, B. Cooper, and B. Wilcox, “Experiences with operations and autonomy of the mars pathfinder microrover,” in *IEEE Aerospace Conference*, 1998. 24

[116] M. Bajracharya, M. Maimone, and D. Helmick, “Autonomy for mars rovers; past, present, and future,” *IEEE Computer*, vol. 41, pp. 44–50, December 2008. 24

[117] L. Matthies, M. Maimone, A. Johnson, Y. Cheng, R. Willson, C. Villalpando, S. Goldberg, A. Huertas, A. Stein, and A. Angelova, “Computer vision on mars,” *International Journal of Computer Vision*, vol. 75, no. 1, pp. 67–92, 2007. 24, 37

[118] M. Maimone, P. C. Leger, and J. Biesiadecki, “Overview of the mars exploration rovers’ autonomous mobility and vision capabilities,” in *Proceedings IEEE International Conference on Robotics and Automation*, 2007. 24

[119] J. Biesiadecki, P. C. Leger, and M. Maimone, “Tradeoffs between directed and autonomous driving on the mars exploration rovers,” *International Journal of Robotics Research*, vol. 26, pp. 91–104, 2007. 24

[120] J. Biesiadecki, M. Maimone, and J. Morrison, “The athena sdm rover: a testbed for mars rover mobility,” in *International Symposium on Artificial Intelligence*, 2001. 24, 30, 35, 90

[121] S. Goldberg, M. Maimone, and L. Matthies, “Stereo vision and rover navigation software for planetary exploration,” *Aerospace Conference Proceedings, 2002. IEEE*, vol. 5, pp. 5–2025–5–2036 vol.5, 2002. 24, 29, 30, 35, 90

[122] J. Biesiadecki and M. Maimone, “The mars exploration rover surface mobility flight software driving ambition,” in *IEEE Aerospace Conference*, 2006. 24, 30, 35, 90

[123] R. Simmons, E. Krotkov, L. Chrisman, F. Cozman, R. Goodwin, M. Hebert, L. Katragadda, S. Koenig, G. Krishnaswamy, Y. Shinoda, W. L. Whittaker, and P. Klarer, “Experience with rover navigation for lunar-like terrains,” in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, pp. 441–446, Aug 1995. 24, 30, 35

[124] S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, and K. Schwehr, “Recent progress in local and global traversability for planetary rovers,” *IEEE Conference on Robotics and Automation*, 2000. 24, 29, 30, 35, 90

[125] C. Urmson, M. B. Dias, and R. Simmons, “Stereo vision based navigation for sun-synchronous exploration,” in *Proceedings of the International Conference on Robotics and Automation*, 2002. 24, 29, 30, 31, 35

[126] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz, “Global path planning on-board the mars exploration rovers,” in *IEEE Aerospace Conference*, 2007. 24, 30

[127] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz, “Global planning on the mars exploration rovers: Software integration and surface testing,” *Journal of Field Robotics*, vol. 26, pp. 337–357, 2009. 24, 30, 90

[128] E. Birgersson, A. Howard, and G. Sukhatme, “Towards stealthy behaviors,” in *International Conference on Intelligent Robots and Systems*, pp. 1703 – 1708, October 2003. 28

[129] R. Kirby, R. Simmons, and J. Forlizzi, “Companion: A constraint-optimizing method for personacceptable navigation,” in *International Symposium on Robot and Human Interactive Communication*, September 2009. 28, 30, 86

[130] Y.-H. Chen, C.-H. Lu, K.-C. Hsu, L.-C. Fu, Y.-J. Yeh, and L.-C. Kuo, “Preference model assisted activity recognition learning in a smart home environment,” in *International Conference on Intelligent Robots and Systems*, pp. 4657 – 4662, October 2009. 28

[131] C. Ye and J. Borenstein, “A method for mobile robot navigation on rough terrain,” in *International Conference on Robotics and Automation*, pp. 3863–3869, 2004. 30

[132] V. Molino, R. Madhavan, E. Messina, A. Downs, S. Balakirsky, and A. Jacoff, “Traversability metrics for rough terrain applied to repeatable test methods,” in *International Conference on Intelligent Robots and Systems*, 2007. 30, 35

[133] A. Chilian and H. Hirschmuller, “Stereo camera based navigation of mobile robots on rough terrain,” in *International Conference on Intelligent Robots and Systems*, pp. 4571–4576, 2009. 30

[134] H. Seraji, “Traversability index: a new concept for planetary rovers,” in *Proceedings IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2006–2013 vol.3, 1999. 30, 35

[135] H. Seraji, “Fuzzy traversability index: A new concept for terrain-based navigation,” *Journal of Robotic Systems*, vol. 17, no. 2, pp. 79–91, 2000. 30, 35

[136] A. Howard and H. Seraji, “An intelligent terrain-based navigation system for planetary rovers,” *IEEE Robotics & Automation Magazine*, vol. 8, pp. 9–17, Dec 2001. 30, 35

[137] A. Howard, E. Tunstel, D. Edwards, and A. Carlson, “Enhancing fuzzy robot navigation systems by mimicking human visual perception of natural terrain traversability,” in *Joint IFSA World Congress and NAFIPS International Conference*, vol. 1, pp. 7–12 vol.1, July 2001. 30, 35, 38

[138] H. Seraji and A. Howard, “Behavior-based robot navigation on challenging terrain: A fuzzy logic approach,” *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 308–321, Jun 2002. 30, 35, 38

[139] H. Seraji and B. Bon, “Multi-range traversability indices for terrain-based navigation,” in *IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2674–2681 vol.3, 2002. 30, 35

[140] E. Tunstel, A. Howard, and H. Seraji, “Rule based reasoning and neural network perception for safe offroad robot mobility,” *Expert Systems*, vol. 19, pp. 191–200, 2002. 30, 35, 37

[141] A. Howard, H. Seraji, and B. Werger, “Global and regional path planners for integrated planning and navigation,” *Journal of Robotic Systems*, vol. 22, pp. 767–778, 2005. 30, 35

[142] D. B. Gennery, “Traversability analysis and path planning for a planetary rover,” *Auton. Robots*, vol. 6, no. 2, pp. 131–146, 1999. 30

[143] C. Thorpe, “Path relaxation: Path planning for a mobile robot,” Tech. Rep. CMU-RI-TR-84-05, Robotics Institute, Pittsburgh, PA, April 1984. 30

[144] X. Ning, S. Shihuang, and F. Xizhou, “A fuzzy approach to the weighted region problem for autonomous vehicles,” in *Proceedings of the International Symposium on Intelligent Control*, 1993. 30

[145] R. Murphy, K. Hughes, and E. Noll, “An explicit path planner to facilitate reactive control and terrain preferences,” in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 3, pp. 2067–2072 vol.3, Apr 1996. 30

[146] J. Rosenblatt, “Utility fusion: Map-based planning in a behavior-based system,” in *Field and Service Robotics*, 1997. 30

[147] A. Shirkhodaie, R. Amrani, N. Chawla, and T. Vicks, “Traversable terrain modeling and performance measurement of mobile robots,” in *PerMIS*, 2004. 30

[148] K. Konolige, M. Agrawal, M. R. Blas, R. C. Bolles, B. Gerkey, J. Sola, and A. Sundaresan, “Mapping, navigation, and learning for off-road traversal,” *Journal of Field Robotics*, vol. 26, pp. 88–113, January 2009. 30, 39

[149] A. Stentz, "CD*: a real-time resolution optimal re-planner for globally constrained problems," in *Proceedings of AAAI National Conference on Artificial Intelligence*, July 2002. 30

[150] J. P. Gonzalez, B. Nagy, and A. Stentz, "The geometric path planner for navigating unmanned vehicles in dynamic environments," in *Proceedings ANS 1st Joint Emergency Preparedness and Response and Robotic and Remote Systems*, February 2006. 30

[151] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation.," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. Rumelhart and J. McClelland, eds.), vol. 1, pp. 318–362., Cambridge, MA: MIT Press, 1986. 31

[152] K. Iagnemma, F. Genot, and S. Dubowsky, "Rapid physics-based rough-terrain rover planning with sensor and control uncertainty," in *IEEE Internation Conference on Robotics and Automation*, 1999. 35

[153] M. Cherif, "Motion planning for all-terrain vehicles: a physical modeling approach for coping with dynamic and contact interaction constraints," *IEEE Transactions on Robotics and Automation*, vol. 15, pp. 202–218, Apr 1999. 35

[154] A. Green and D. Rye, "Sensible planning for vehicles operating over difficult unstructured terrains," *IEEE Aerospace Conf.*, March 2007. 35

[155] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007. 35, 89

[156] D. Helmick, A. Angelova, and L. Matthies, "Terrain adaptive navigation for planetary rovers," *Journal of Field Robotics*, vol. 26, pp. 391–410, 2009. 35, 40

[157] D. Pomerleau, "Alvinn: an autonomous land vehicle in a neural network," *Advances in neural information processing systems 1*, pp. 305 – 313, 1989. 36, 42

[158] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, "Off-road obstacle avoidance through end-to-end learning," in *Advances in Neural Information Processing Systems 18*, MIT Press, 2006. 36, 42

[159] M. Marra, R. Dunlay, and D. Mathis, "Terrain classification using texture for the alv," in *Proceedings of SPIE*, vol. 1007, pp. 64–70, 1988. 37, 153

[160] I. Davis, A. Kelly, A. Stentz, and L. Matthies, "Terrain typing for real robots," in *Proceedings of IEEE Intelligent Vehicles Conference*, pp. 400 – 405, September 1995. 37, 43, 153

[161] C. Rasmussen, "Combining laser range, color, and texture cues for autonomous road following," in *IEEE Conference on Robotics and Automation*, 2002. 37

[162] P. Jansen, W. van der Mark, J. van den Heuvel, and F. Groen, "Colour based off-road environment and terrain type classification," in *Proceedings IEEE Intelligent Transportation Systems*, pp. 216–221, 2005. 37

[163] A. Angelova, L. Matthies, D. Helmick, and P. Perona, "Learning and prediction of slip from visual information," *Journal of Field Robotics*, vol. 24, no. 3, pp. 205–231, 2007. 37, 40

[164] I. Halatci, C. Brooks, and K. Iagnemma, "Terrain classification and classifier fusion for planetary exploration rovers," in *IEEE Aerospace Conference*, March 2007. 37

[165] R. Karlsen and G. Witus, "Terrain understanding for robot navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 895–900, 2007. 37

[166] A. P. Charaniya, R. Manduchi, and S. K. Lodha, "Supervised parametric classification of aerial lidar data," in *IEEE Conference on Computer Vision and Pattern Recognition Workshop*, pp. 25–32, 2004. 37

[167] D. Sadhukhan, C. Moore, and E. Collins, "Terrain estimation using internal sensors," in *IASTED Conference on Robotic Applications*, 2004. 37

[168] C. Brooks, K. Iagnemma, and S. Dubowsky, "Vibration-based terrain analysis for mobile robots," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 3415–3420, April 2005. 37

[169] C. Brooks and K. Iagnemma, "Vibration-based terrain classification for planetary exploration rovers," *IEEE Transactions on Robotics*, vol. 21, pp. 1185–1191, Dec. 2005. 37

[170] L. Ojeda, J. Borenstein, G. Witus, and R. Karlsen, "Terrain characterization and classification with a mobile robot," *Journal of Field Robotics*, vol. 23, no. 2, pp. 103–122, 2006. 37, 40

[171] M. Happold, M. Ollis, and N. Johnson, "Enhancing supervised terrain classification with predictive unsupervised learning," in *Proceedings of Robotics: Science and Systems*, (Philadelphia, USA), August 2006. 37, 39, 41

[172] M. Happold and M. Ollis, "Using learned features from 3d data for robot navigation," *Autonomous Robots and Agents*, vol. 76, pp. 61–69, 2007. 37

[173] W. H. Huang, M. Ollis, M. Happold, and B. A. Stancil, "Image-based path planning for outdoor mobile robots," *Journal of Field Robotics*, vol. 26, pp. 196–211, 2009. 37

[174] A. Howard, M. Turmon, L. Matthies, B. Tang, A. Angelova, and E. Mjolsness, "Towards learned traversability for robot navigation: From underfoot to the far field," *Journal of Field Robotics*, vol. 23, pp. 1005–1017, 2007. 37, 39

[175] S. Thrun, M. Montemerlo, and A. Aron, "Probabilistic terrain analysis for high-speed desert driving," in *Proceedings of Robotics: Science and Systems*, (Philadelphia, USA), August 2006. 37

[176] J. Sun, T. Mehta, D. Wooden, M. Powers, J. Rehg, T. Balch, and M. Egerstedt, "Learning from examples in unstructured, outdoor environments," *Journal of Field Robotics*, vol. 23, pp. 1019–1036, 2007. 37, 38, 43, 53, 85

[177] M. Ollis, W. H. Huang, and M. Happold, “A bayesian approach to imitation learning for robot navigation,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007. 38

[178] L. Jackel, E. Krotkov, M. Perschbacher, J. Pippine, and C. Sullivan, “The darpa lagr program: Goals, challenges, methodology, and phase i results,” *Journal of Field Robotics*, vol. 23, pp. 945–973, 2007. 38

[179] D. Lieb, A. Lookingbill, and S. Thrun, “Adaptive road following using self-supervised learning and reverse optical flow,” in *Proceedings of Robotics: Science and Systems*, (Cambridge, USA), June 2005. 39

[180] M. Shneier, T. Chang, T. Hong, W. Shackleford, R. Bostelman, and J. Albus, “Learning traversability models for autonomous mobile vehicles,” *Autonomous Robots*, vol. 24, pp. 69–86, 2008. 39, 40

[181] J. Albus, R. Bostelman, T. Chang, T. Hong, W. Shackleford, and M. Shneier, “Learning in a hierarchical control system: 4d/rcs in the darpa lagr program,” *Journal of Field Robotics*, vol. 23, pp. 975–1003, 2007. 39, 40

[182] P. Vernaza, B. Taskar, and D. Lee, “Online, self-supervised terrain classification via discriminatively trained submodular markov random fields,” in *IEEE International Conference on Robotics and Automation*, pp. 2750–2757, May 2008. 39

[183] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun, “Learning long-range vision for autonomous off-road driving,” *Journal of Field Robotics*, vol. 26, pp. 120–144, 2009. 39

[184] P. Moghadam and W. Wijesoma, “Online, self-supervised vision-based terrain classification in unstructured environments,” in *International Conference on Systems, Man and Cybernetics*, pp. 3100–3105, October 2009. 39

[185] M. Bajracharya, B. Tang, A. Howard, M. Turmon, and L. Matthies, “Learning long-range terrain classification for autonomous navigation,” in *IEEE International Conference on Robotics and Automation*, pp. 4018–4024, May 2008. 39

[186] G. Grudic and J. Mulligan, “Outdoor path labeling using polynomial mahalanobis distance,” in *Proceedings of Robotics: Science and Systems*, (Philadelphia, USA), August 2006. 39

[187] M. Procopio, J. Mulligan, and G. Grudic, “Long-term learning using multiple models for outdoor autonomous robot navigation,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3158–3165, 2007. 39

[188] D. Kim, J. Sun, S. M. Oh, J. Rehg, and A. Bobick, “Traversability classification using unsupervised on-line visual learning for outdoor robot navigation,” in *Proceedings IEEE International Conference on Robotics and Automation*, pp. 518–525, May 2006. 40

[189] D. Kim, S. M. Oh, and J. Rehg, “Traversability classification for ugv navigation: a comparison of patch and superpixel representations,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3166–3173, 2007. 40

[190] D. Stavens and S. Thrun, “A self-supervised terrain roughness estimator for off-road autonomous driving,” in *In Proc. of Conf. on Uncertainty in AI*, pp. 13–16, 2006. 40

[191] A. Angelova, L. Matthies, D. Helmick, G. Sibley, and P. Perona, “Learning to predict slip for ground robots,” in *Proceedings IEEE International Conference on Robotics and Automation*, May 2006. 40

[192] S. Karumanchi, T. Allen, T. Bailey, and S. Scheding, “Non-parametric learning to aid path planning over slopes,” in *Robotics: Science and Systems*, June 2009. 40, 154

[193] S. Karumanchi, T. Allen, T. Bailey, and S. Scheding, “Non-parametric learning to aid path planning over slopes,” *International Journal of Robotics Research*, vol. 29, pp. 997–1018, July 2010. 40, 154

[194] K. Iagnemma, S. Kang, H. Shibly, and S. Dubowsky, “Online terrain parameter estimation for wheeled mobile robots with application to planetary rovers,” *IEEE Transactions on Robotics*, vol. 20, pp. 921–927, Oct. 2004. 40

[195] C. Wellington and A. Stentz, “Learning predictions of the load-bearing surface for autonomous rough-terrain navigation in vegetation,” in *International Conference on Field and Service Robotics*, pp. 49–54, July 2003. 40

[196] C. Wellington and A. Stentz, “Online adaptive rough-terrain navigation vegetation,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004. 40

[197] C. Wellington, A. Courville, and A. Stentz, “A generative model of terrain for autonomous navigation in vegetation,” *The International Journal of Robotics Research*, vol. 25, pp. 1287 – 1304, December 2006. 40

[198] R. Sutton, A. Barto, and R. Williams, “Reinforcement learning is direct adaptive optimal control,” *IEEE Control Systems Magazine*, vol. 12, pp. 19–22, April 1992. 41

[199] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, “An application of reinforcement learning to aerobatic helicopter flight,” in *Neural Information Processing Systems*, 2007. 41

[200] C. Gaskett, L. Fletcher, and A. Zelinsky, “Reinforcement learning for a vision based mobile robot,” in *IEEE/RSJ Conference on Intelligent Robots and Systems*, 2000. 41

[201] K. Macek, I. Petrovic, and N. Peric, “A reinforcement learning approach to obstacle avoidance of mobile robots,” in *Advanced Motion Control*, 2002. 41, 42

[202] W. D. Smart and L. Pack Kaelbling, “Effective reinforcement learning for mobile robots,” in *Proc. IEEE International Conference on Robotics and Automation ICRA ’02*, vol. 4, pp. 3404–3410, 11–15 May 2002. 41, 42

[203] J. Michels, A. Saxena, and A. Y. Ng, "High speed obstacle avoidance using monocular vision and reinforcement learning," in *International Conference on Machine Learning*, vol. 119, pp. 593–600, 2005. 41

[204] S. Schaal and C. Atkeson, "Open loop stable control strategies for robot juggling," in *Proceedings of the 93 IEEE Int. Conf. on Robotics and Automation*, 1993. 42

[205] S. Schaal and C. Atkeson, "Robot juggling: An implementation of memory-based learning," *IEEE Control Systems Magazine*, vol. 14, no. 1, 1994. 42

[206] B. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, 2008. 42

[207] C. Sammut, S. Hurst, D. Kedzier, and D. Michie, "Learning to fly," in *International Conference on Machine Learning*, 1992. 42

[208] N. Esmaili, C. Sammut, and G. Shirazi, "Behavioural cloning in control of a dynamic system," in *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, vol. 3, pp. 2904–2909 vol.3, Oct 1995. 42

[209] M. W. Kadous, C. Sammut, and R. Sheh, "Autonomous traversal of rough terrain using behavioural cloning," in *International Conference on Autonomous Robots and Automation*, 2006. 42, 85

[210] A. Howard, B. Werger, and H. Seraji, "A human-robot mentor-protege relationship to learn off-road navigation behavior," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 430–435 Vol. 1, Oct. 2005. 42

[211] I. Davis and A. Stentz, "Sensor fusion for autonomous outdoor navigation using neural networks," in *Proceedings IEEE/RSJ International Conference On Intelligent Robotic Systems*, vol. 3, pp. 338 – 343, August 1995. 42

[212] J. Weng and S. Chen, "Autonomous navigation through case-based learning," in *Proceedings International Symposium on Computer Vision*, pp. 359–364, Nov 1995. 43

[213] M. Stolle, H. Tappeiner, J. Chestnutt, and C. Atkeson, "Transfer of policies based on trajectory libraries," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2981–2986, 2007. 43

[214] B. Argall, B. Browning, and M. Veloso, "Learning by demonstration with critique from a human teacher," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, (New York, NY, USA), pp. 57–64, ACM, 2007. 43

[215] P. Sermanet, M. Scoffier, C. Crudele, U. Muller, and Y. LeCun, "Learning maneuver dictionaries for ground robot planning," in *International Symposium on Robotics*, 2008. 43

[216] R. Roberts, C. Pippin, and T. Balch, "Learning outdoor mobile robot behaviors by example," *Journal of Field Robotics*, vol. 26, no. 2, pp. 176 – 195, 2009. 43

[217] B. Hamner, S. Scherer, and S. Singh, "Learning to drive among obstacles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2663 – 2669, October 2006. 43, 85

[218] B. Hamner, S. Singh, and S. Scherer, "Learning obstacle avoidance parameters from operator behavior," *Journal of Field Robotics*, vol. 23, pp. 1037–1058, December 2006. 43, 85

[219] R. Brooks, "Intelligence without representation," *Artificial Intelligence*, vol. 47, pp. 139–159, 1991. 43

[220] R. Kalman, "When is a linear control system optimal?," *Trans. ASME, J. Basic Engrg.*, vol. 86, pp. 51–60, 1964. 43

[221] S. Boyd, L. E. Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Society for Industrial and Applied Mathematics (SIAM), 1994. 43

[222] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proc. 17th International Conf. on Machine Learning*, 2000. 45

[223] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *International Conference on Machine learning*, 2004. 45, 50

[224] N. Ratliff, J. A. Bagnell, and M. Zinkevich, "Maximum margin planning," in *International Conference on Machine Learning*, July 2006. 45

[225] N. Ratliff, D. Bradley, J. A. Bagnell, and J. Chestnutt, "Boosting structured prediction for imitation learning," in *Advances in Neural Information Processing Systems 19*, (Cambridge, MA), MIT Press, 2007. 45, 55, 68

[226] J. A. Bagnell, J. Langford, N. Ratliff, and D. Silver, "The exponentiated functional gradient algorithm for structured prediction problems," in *The Learning Workshop*, 2007. 45, 53

[227] N. Ratliff, S. Srinivasa, and J. A. Bagnell, "Imitation learning for locomotion and manipulation," in *IEEE-RAS International Conference on Humanoid Robots*, November 2007. 45

[228] N. Ratliff, J. A. Bagnell, and M. Zinkevich, "(Online) subgradient methods for structured prediction," in *Artificial Intelligence and Statistics*, (San Juan, Puerto Rico), 2007. 45, 47

[229] D. Silver, J. A. Bagnell, and A. Stentz, "High performance outdoor navigation from overhead data using imitation learning," in *Proceedings of Robotics Science and Systems*, June 2008. 45, 53, 56, 59, 68

[230] D. Silver, J. A. Bagnell, and A. Stentz, "Applied imitation learning for autonomous navigation in complex natural terrain," in *Field and Service Robotics*, July 2009. 45, 68

[231] D. Silver, J. A. Bagnell, and A. Stentz, "Perceptual interpretation for autonomous navigation through dynamic imitation learning," in *International Symposium on Robotics Research*, August 2009. 45, 59, 152

[232] D. Silver, J. A. Bagnell, and A. Stentz, "Learning from demonstration for autonomous navigation in complex unstructured terrain," *International Journal of Robotics Research*, vol. 29, pp. 1565–1592, October 2010. 45, 46, 53, 59

[233] B. Taskar, S. Lacoste-Julien, and M. Jordan, "Structured prediction via the extragradient method," in *Advances in Neural Information Processing Systems 18*, MIT Press, 2006. 47

[234] B. Taskar, C. Guestrin, and D. Koller, "Max-margin markov networks," in *Advances in Neural Information Processing Systems*, 2004. 47

[235] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent," in *Advances in Neural Information Processing Systems 12*, (Cambridge, MA), MIT Press, 2000. 51

[236] B. Nabbe, S. Kumar, and M. Hebert, "Path planning with hallucinated worlds," in *Proceedings: IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2004. 56

[237] A. Karmaker and S. Kwek, "A boosting approach to remove class label noise," in *International Conference on Hybrid Intelligent Systems*, 2005. 66

[238] A. Vezhnevets and O. Barinova, "Avoiding boosting overfitting by removing confusing samples," in *European Conference on Machine Learning*, 2007. 66

[239] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The field D* algorithm," *Journal of Field Robotics*, vol. 23, pp. 79–101, February 2006. 69, 72

[240] B. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *IROS*, 2009. 71, 136

[241] A. Coates, P. Abbeel, and A. Y. Ng, "Learning for control from multiple demonstrations," in *International Conference on Machine Learning*, pp. 144–151, 2008. 85

[242] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *International Journal of Robotics Research (OnlineFirst)*, vol. 29, pp. 1608–1639, November 2010. 85

[243] P. Abbeel, D. Dolgov, A. Y. Ng, and S. Thrun, "Apprenticeship learning for motion planning with application to parking lot navigation," in *International Conference on Intelligent Robots and Systems*, 2008. 85

[244] P. Henry, C. Vollmer, B. Ferris, and D. Fox, "Learning to navigate through crowded environments," in *International Conference on Robotics and Automation*, 2010. 85

[245] S. J. Lee and Z. Popovic, "Learning behavior styles with inverse reinforcement learning," in *SIGGRAPH*, vol. 29, July 2010. 85

[246] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, pp. 308–333, March 2009. 89

[247] T. Howard, C. J. Green, and A. Kelly, "Receding horizon model-predictive control for mobile robot navigation of intricate paths," in *Field and Service Robotics*, 2009. 90

[248] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *Journal of Machine Learning Research*, vol. 6, pp. 1453–1484, 2005. 94

[249] W. W. Cohen, R. E. Schapire, and Y. Singer, "Learning to order things," *Journal of Artificial Intelligence Research*, vol. 10, pp. 243–270, May 1999. 127

[250] R. Herbrich, T. Graepel, and K. Obermayer, "Support vector learning for ordinal regression," in *International Conference on Artificial Neural Networks*, 1999. 127

[251] T. Joachims, "Optimizing search engines using clickthrough data," in *ACM Conference on Knowledge Discovery and Data Mining*, 2002. 127

[252] M. Zucker, J. A. Bagnell, C. Atkeson, and J. Kuffner, "An optimization approach to rough terrain locomotion," in *IEEE Conference on Robotics and Automation*, May 2010. 128, 153

[253] M. Kalakrishnan, J. Buchli, P. Pastor, and S. Schaal, "Learning locomotion over rough terrain using terrain templates," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 167–172, 2009. 128, 153

[254] J. Z. Kolter, P. Abbeel, and A. Y. Ng, "Hierarchical apprenticeship learning with application to quadruped locomotion," in *Neural Information Processing Systems*, 2008. 128, 153

[255] M. Zucker and J. A. Bagnell, "Reinforcement planning: Rl for optimal planners," Tech. Rep. CMU-RI-TR-10-14, Carnegie Mellon Robotics Institute, April 2010. 133

[256] D. Roth and K. Small, "Margin based active learning for structured output spaces," in *ECML*, 2006. 134

[257] B. Settles and M. Craven, "An analysis of active learning strategies for sequence labeling tasks," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2008. 134

[258] A. Culotta and A. McCallum, "Reducing labeling effort for structured prediction tasks," in *AAAI*, 2005. 134

[259] B. Settles, M. Craven, and L. Friedland, "Active learning with real annotation costs," in *NIPS Workshop on Cost Sensitive Learning*, 2008. 134

[260] R. Haertel, K. D. Seppi, E. K. Ringger, and J. L. Carroll, "Return on investment for active learning," in *NIPS Workshop on Cost Sensitive Learning*, 2009. 134

[261] S. Chernova and M. Veloso, "Confidence-based policy learning from demonstration using gaussian mixture models," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007. 134

[262] D. Grollman and O. Jenkins, "Dogged learning for robots," in *IEEE International Conference on Robotics and Automation*, 2007. 134

[263] M. Markou and S. Singh, "Novelty detection: A review - part 1: Statistical approaches," *Signal Processing*, vol. 83, no. 12, pp. 2481–2497, 2003. 135

[264] M. Markou and S. Singh, "Novelty detection: A review - part 2: Neural network based approaches," *Signal Processing*, vol. 83, no. 12, pp. 2499–2521, 2003. 135

[265] H. Neto and U. Nehmzow, "Visual novelty detection with automatic scale selection," *Robotics and Autonomous Systems*, vol. 55, no. 9, pp. 693–701, 2007. 135

[266] S. Marsland, U. Nehmzow, and J. Shapiro, "On-line novelty detection for autonomous mobile robots," *Robotics and Autonomous Systems*, vol. 51, no. 2-3, pp. 191–206, 2005. 135

[267] C. Brooks and K. Iagnemma, "Visual detection of novel terrain via two-class classification," in *ACM symposium of Applied Computing*, pp. 1145–1150, 2009. 135, 136

[268] D. R. Thompson, "Domain-guided novelty detection for autonomous exploration," in *International Joint Conference on Artificial Intelligence*, 2009. 135

[269] B. Sofman, J. A. Bagnell, and A. Stentz, "Anytime online novelty detection for vehicle safeguarding," in *IEEE International Conference on Robotics and Automation*, May 2010. 135, 136, 147

[270] C. Dima, M. Hebert, and A. Stentz, "Enabling learning from large datasets: Applying active learning to mobile robotics," in *International Conference on Robotics and Automation*, vol. 1, pp. 108 – 114, April 2004. 135

[271] C. Dima and M. Hebert, "Active learning for outdoor obstacle detection," in *Proceedings of Robotics: Science and Systems*, (Cambridge, USA), June 2005. 135, 137

[272] I. Fodor, "A survey of dimension reduction techniques," Tech. Rep. UCRL-ID-148494, Lawrence Livermore National Laboratory, May 2002. 135

[273] D. Lewis and W. Gale, "A sequential algorithm for training text classifiers," in *Proceedings of the International ACM-SIGIR Conference on Research and Development in Information Retrieval*, 1994. 137

[274] H. Seung, M. Opper, and H. Sompolinsky, "Query by committee," in *Proceedings of the Fifth Annual ACM Workshop on Computation Learning Theory*, pp. 287–294, 1992. 137

[275] Y. Freund, H. Seung, E. Shamir, and N. Tishby, "Selective sampling using the query by committee algorithm," *Machine Learning*, vol. 28, pp. 133–168, 1997. 137

[276] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996. 137

[277] L. Murphy and P. Newman, "Planning most-likely paths from overhead imagery," in *International Conference on Robotics and Automation*, 2010. 138

[278] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967. 143

[279] D. Bradley, *Learning in Modular Systems*. PhD thesis, Carnegie Mellon University, September 2009. 153

[280] A. Grubb and J. A. Bagnell, “Boosted backpropagation learning for training deep modular networks,” in *International Conference on Machine Learning*, 2010. 153

[281] R. Sheh, M. Kadous, C. Sammut, and B. Hengst, “Extracting terrain features from range images for autonomous random stepfield traversal,” in *IEEE International Workshop on Safety, Security, and Rescue Robotics*, 2007. 153

Author Index